Faculdade de Engenharia da Universidade do Porto

Automação Industrial com recurso a ferramentas "Open Source"

José Ricardo Ribeiro Gomes

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Eletrotécnica e de Computadores
Major Automação

Orientador: Mário Jorge Rodriges de Sousa (Professor Doutor)

Junho 2014

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES 2013/2014

A Dissertação intitulada

"Automação Industrial com Recurso a Ferramentas "Open Source""

foi aprovada em provas realizadas em 24-07-2014

o júri

Presidente Professor Doutor Rui Manuel Esteves Araújo
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Jaime Francisco Cuz Fonseca
Professor Associado do Departamento de Electrónica Industrial da Escola de Engenharia da Universidade do Minho

Professor Doutor Mário Jorge Rodrigues de Sousa
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Autor - José Ricardo Ribeiro Gomes

Resumo

Actualmente, a automação industrial tem uma grande importância nas mais diversas unidades industriais espalhadas pelos diferentes sectores económicos.

Nesta dissertação, é realizado um estudo sobre ferramentas *open source* capazes de criar programas de automação segundo a norma IEC 61131 e ferramentas *open source* para a criação de programas SCADA. Além de uma prospecção de mercado exaustiva, na procura de quais as soluções já existentes e qual o seu estado de desenvolvimento, experimentaram-se alguns dos *softwares* encontrados de maneira a comprovar as suas capacidades. Para a criação de ferramentas SCADA há uma maior oferta. Assim, testaram-se três destes *softwares*: *ScadaBR*, *PVBrowser* e E*clipse SCADA*. Quanto à automatização de processos, a oferta é limitada, sendo que apenas um *software* foi testado, o *Beremiz*. Relativamente a este *software*, realizaram-se inúmeros testes que avaliaram a sua facilidade de utilização, fiabilidade, robustez e conformidade com a norma IEC 61131-3.

Para o teste e validação destas mesmas ferramentas, desenvolveu-se um programa no *software Beremiz* para controlar o kit "Linha de Produção Flexível" existente na FEUP e um programa SCADA, realizado no *ScadaBR*, para a supervisão do mesmo.

Abstract

Currently, industrial automation has a great importance in several industrial units spread across different economic sectors.

In this dissertation, a study on open source tools capable of creating automation programs according to IEC 61131 standard and open source tools for creating SCADA programs is performed. In addition to a thorough market survey, in the demand for what tools already exist and what is their state of development. Some of that software was tested to verify their capabilities. To create SCADA programs there is a big amount of programs available. Thus, three of these software's were tested: *ScadaBR*, *PVBrowser* and *Eclipse SCADA*. As for process automation, supply is much smaller, and only one software has been tested, *Beremiz*. For this software, there have been numerous tests that evaluated its ease of use, reliability, robustness and compliance with IEC 61131-3.

For the test and validation of these same tools, it was developed a program in *Beremiz* software to control the "Flexible Production Line" at FEUP and a SCADA program, made with *ScadaBR*, for the supervision of the same kit.

Agradecimentos

Ao meu orientador, professor Mário Jorge Rodrigues de Sousa, por toda a disponibilidade demonstrada ao longo da elaboração desta dissertação e pelo auxílio dado. Sem a sua ajuda, a conclusão da mesma não teria sido possível.

A todos aqueles que directa ou indirectamente contribuíram para a realização deste trabalho.

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

Lista de abreviaturas

FEUP	Faculdade de Engenharia da Universidade do Porto
SCADA	Supervisory Control And Data Acquisition
OSI	Open Source Iniciative
DFSG	Debian Free Software Guidelines
CPU	Central Processing Unit
PLC	Programmable Logic Controllers
IEC	International Electrotechnical Comission
IDE	Integrated Development Environment
ST	Structured Text
IL	Instruction List
FBD	Function Block Diagram
LD	Ladder Diagram
SFC	Sequential Function Chart
CFC	Continuous Function Chart
POU	Program Organization Units
ADU	Application Data Unit
PDU	Protocol Data Unit
FB	Function Block
RTU	Remote Terminal Units
HMI	Human Machine Interface
GNU	GNU is Not Unix
GPL	General Public License
LGPL	Lesser General Public License
AGLP	Atheros General Public License
BSD	Berkeley Software Distribution
FSF	Free Software Foundation
OS (SO)	Operative System (Sistema Operativo)

ASP	Application Server Provider
SaaS	Software as a service
DRM	Digital Rights Management
MIT	Massachussets Institute of Technology
AFL	Academic Free License
LPF	Linha de Produção Flexível
URL	Uniform Resource Locator
Daemon	Disk And Execution MONitor
M2M	Machine to Machine
RAM	Random Access Memory
Mutex	Mutual Exclusion
fifo	First In First Out
EPL	Eclipse Public License
API	Application Programming Interface
SOAP	Simple Object Access Control

Lista de símbolos

^{a}C	Graus Celsius
s	segundos
ms	milisegundos

Capítulo 1

Introdução

1.1 Contexto da Dissertação

O termo automação pode ser definido como um processo onde são realizadas várias operações industriais com o auxílio de diversos dispositivos eletrónicos e/ou mecânicos que controlam os seus próprios processos. Ao longo dos últimos anos, os sistemas de automação foram evoluindo desde sistemas baseados em controlo automático, mecanizado (com as primeiras linhas de montagem do século XX) até aos sistemas baseados nas actuais tecnologias como a microelectrónica [1]. Tudo isto foi surgindo devido às necessidades sentidas em busca do aumento da produtividade e, consequentemente, melhores resultados a nível de produção. Como consequência, as máquinas foram-se tornando cada vez mais modernas, mais precisas e mais completas, pondo o trabalho humano de parte.

Automação industrial é a aplicação de técnicas, *softwares* e/ou equipamentos específicos numa determinada máquina ou processo industrial, com o objetivo de aumentar a sua eficiência e maximizar a produção com o menor consumo de energia e/ou matérias [2]. O primeiro esforço real para a padronização das linguagens de programação para a automação industrial foi a norma IEC 61131-3, publicada pela Comissão Electrónica Internacional (IEC), que define um conjunto de normas e especificações técnicas com o objectivo de standardizar os autómatos programáveis.

Para esta dissertação, para além da automação industrial, o conceito de *open source* é também fundamental. Resumidamente, para um *software* ser considerado de código aberto a sua licença não deve restringir de maneira alguma a venda ou distribuição do programa gratuitamente. O código fonte do programa tem que ser fornecido ao utilizador e este pode modificá-lo e redistribui-lo.

1.2 Objectivos da Dissertação

O objectivo desta dissertação é o de avaliar o estado actual das ferramentas livres disponíveis para projetos de automação industrial. Realizou-se um estudo sobre as plataformas existentes para o desenvolvimento de interfaces do tipo SCADA, bem como sobre ferramentas para a automação de processos.

O projecto consiste na avaliação dessas mesmas ferramentas quanto à facilidade de utilização, fiabilidade, robustez e conformidade com a norma IEC 61131-3. Para o teste e validação das ferramentas, utilizou-se uma plataforma de simulação da linha de montagem existente num laboratório da FEUP para a qual se desenvolveu um programa com o objectivo de automatizar a plataforma e um programa SCADA para a supervisão do mesmo dispositivo.

A ferramenta de automatização de processos, sobre a qual esta dissertação se debruça é o *Beremiz* que é um ambiente de desenvolvimento integrado (IDE) para o desenvolvimento de programas de automação em conformidade com o estabelecido na norma IEC 61131-3. O código escrito pelo utilizador é então convertido em código ANSI-C, possibilitando a sua execução nas mais variadas plataformas.

As ferramentas *ScadaBR*, *Eclipse SCADA* e *PVBrowser* são as ferramentas para a supervisão da plataforma avaliadas nesta dissertação.

1.3 Interesse da Dissertação

O interesse desta dissertação passa por avaliar se as ferramentas *open source* disponíveis actualmente para o desenvolvimento de projectos de automação industrial têm potencial para serem utilizadas no mundo industrial.

Após se proceder à selecção das ferramentas livres para o desenvolvimento de interfaces SCADA e das ferramentas livres disponíveis para a automatização de processos com melhores características, averiguou-se se essas ferramentas são tão capazes e confiáveis como as ferramentas de *software* proprietário. Tendo as mesmas portencialidades que as ferramentas utilizadas actualmente no mundo industrial, as empresas e os utilizadores poderão reduzir exponencialmente os custos que têm com *hardware*/*software*. Isto permitirá também que estudantes consigam praticar a programação segundo a norma IEC-61131-3 sem qualquer custo associado.

Capítulo 2

Documentação Técnica

Neste capítulo é realizado um *technical overview* que aborda alguns aspectos mais técnicos fundamentais para a realização desta dissertação. Assim, elaboraram-se quatro subcapítulos para uma visão mais aprofundada. O primeiro subcapítulo é sobre a norma IEC61131, mais especificamente a sua terceira parte; no segundo subcapítulo explica-se em que consiste um sistema SCADA. O terceiro e o quarto subcapítulos são sobre Licenças de *software* e Protocolo *Modbus*, respectivamente.

2.1 Norma Standard IEC 61131

A norma IEC 61131 estabelece um conjunto de regras que todos os PLC's devem seguir, desde características mecânicas e elétricas a aspectos lógicos e de comunicações. A parte 3 (61131-3 – parte 3 de 8) foi publicada pela 1ª vez em 1993, sendo que a edição actual foi publicada em Fevereiro de 2013 [3]. As grandes vantagens da sua utilização são a uniformização da estrutura funcional do equipamento, bem como a uniformização das linguagens de programação, definindo a sua semântica e sintaxe, isto é, um modelo único para a Engenharia de Software, independente do fabricante.

No seu modelo de programação, especificam-se: tipos de dados (numéricos inteiros, numéricos reais, Booleanos, Strings, Tempos e Datas e ainda tipos de dados derivados), identificadores, Unidades de Organização de Programas (POU) que incluem *Functions*, *Function Blocks* e *Programs*, elementos dos *Sequential Function Charts* (para definição de máquinas de estados, baseado em *Grafcet*) e elementos das configurações (variáveis globais, recursos, tarefas) [4] [5].

Relativamente às linguaguens de programação, a linguagem gráfica *Ladder Diagram* e a linguagem textual *Instruction List* são linguagens muito limitadas na construção de programas complexos. Ainda assim, a linguagem LD é ainda muito utilizada na indústria.

A linguagem ST (*Structured Text*) já se trata de uma linguagem de alto nível parecida com *Pascal*. Esta permite efectuar estruturas (ciclos *while*, *for*, etc) assim como expressões complexas (+, *, AND, OR, etc) e funções (SQRT(), SIN(), etc.) [4] [5].

A linguagem FBD (*Function Block Diagram)* é uma linguagem gráfica que permite descrever um processo através de um conjunto de blocos interligados. Estes blocos representam uma determinada função e a eles estão associadas variáveis de entrada e variáveis de saída. Aas variáveis são conectadas aos blocos através de *connection lines*, sendo que deve haver sempre compatibilização de dados nas ligações, isto é, não se pode ligar uma saída do tipo INT a uma entrada do tipo BOOL. Trata-se portanto de uma linguagem baseada no conceito de fluxo de sinal, podendo por vezes ser problemática quando a correcção do programa depende da sequência de execução dos blocos [4] [5].

A linguagem SFC (*Sequencial Function Chart*) é uma linguagem de definição de máquina de estados que escreve sequências de operações e iterações entre processos paralelos, sequenciais e concorrentes. Há três conceitos fundamentais neste tipo de linguagem que são: transições, etapas e acções. A cada etapa podem ser associadas acções, definidas utilizando qualquer uma das cinco linguagens referidas anteriormente (LD, FBD, ST, IL, SFC), sendo que a cada acção deve estar associado um qualificador (N, P1, P0, R, L, entre outros) que define quando é que a acção deve ser executada. Para haver transição de um estado para o outro é necessário que se verifique a condição da transição. Para cada etapa existem duas variáveis automáticas pré-definidas, <nome_etapa>.X (BOOL) que indica se uma determinada etapa está activa ou não (*TRUE* se activa, *FALSE* se inactiva), e <nome_etapa>.T (TIME) que indica o tempo em que uma etapa está activa, que podem ser utlizadas tanto nas acções como nas transições.

O código desta linguagem pode e deve ser organizado em blocos, potencialmente reutilizáveis. Os tipos de blocos disponíveis são:

- *Program*: O programa é a estrutura de mais alto nível, quase equivalente ao *main()* de um programa em C ou C++. Pode ser escrito em qualquer uma das linguagens: LD, IL, FBD, ST e SFC.
- *Function*: A função permite agrupar código que se pretende executar em vários pontos do programa, sem ter de o repetir, é portanto equivalente às funções de C e Pascal. As funções podem ser invocadas por qualquer outro POU, funções inclusive, e podem ser escritas em LD, IL, FBD e ST. Na definição de uma função esta pode ter parâmetros de entrada, saída, e entrada_saída. Existem várias funções aritméticas, numéricas, lógicas, entre outras já pré-definidas, conforme o definido na norma.
- *Function Block*: Tal como a *function*, o *function block* permite agrupar código que se pretende executar em vários pontos do programa, sem ter de o repetir. A diferença entre uma função e um FB é que um FB pode memorizar resultados de

invocações anteriores. Quando se deseja invocar um FB, é preciso invocar instâncias do mesmo, não podendo um FB ser invocado directamente. Os *function blocks* podem ter parâmetros de entrada, saída e entrada_saída, assim como variáveis internas. Estes blocos podem ser invocados/chamados a partir de programas ou *function blocks* mas não de funções. Assim como nas funções, a norma prevê que com o compilador sejam fornecidos vários FB pré-definidos, que poderão ser usados nos programas [4][5].

2.2 SCADA (Supervisory Control and Data Acquisition)

Os sistemas SCADA, como o seu acrónimo indica, são sistemas desenvolvidos com o objectivo de supervisionar e monitorizar as variáveis dos equipamentos ou plataformas industriais que se pretende controlar. É consituído por:

- Um ou mais dispositivos de interface de dados, geralmente RTUs, ou PLCs, ligados aos sensores do processo, que convertem os sinais analógicos recebidos dos mesmos em sinais digitais entendidos pelo protocolo de comunicação que se esteja a utilizar. Estes sinais são depois enviados para o sistema supervisório.
- Um sistema de comunicações usado para transferir dados entre os dispositivos de campo (RTUs ou PLCs) e o sistema supervisório.
- Um sistema de supervisão que faz a aquisição dos dados do processo e envia comandos de controlo para o mesmo.
- Uma interface Homem-Máquina (HMI) que permite a supervisão e controlo do processo em tempo real pelo operador humano [6]

Na figura 1, é visível a arquitectura de *hardware* deste tipo de sistemas de controlo industrial.

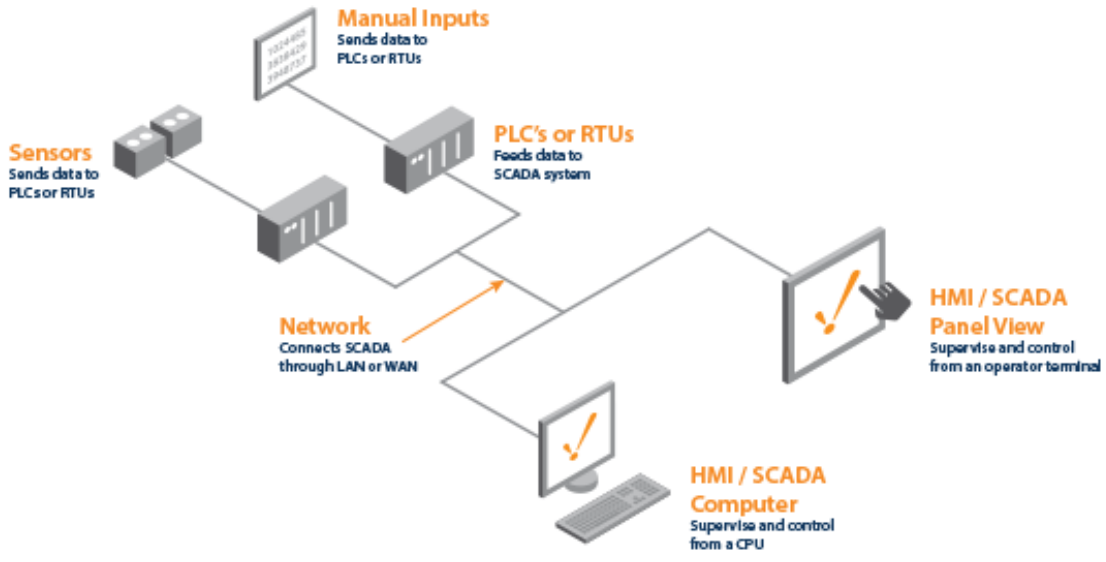

Figura 1 - Hardware típico de um sistema SCADA [7]

Um sistema SCADA deve ser capaz de:

- Fazer a aquisição de variáveis de diferentes plataformas externas através de vários protocolos de comunicação, como o *Ethernet* e o *Modbus*, por exemplo.
- Apresentar dados em tempo real para a monitorização do processo;
- Fornecer interfaces gráficas (exemplificadas na Figura 2) que apresentem o estado actual do processo, permitindo assim uma fácil visualização e análise das variáveis do processo e dos alarmes. Permite também interação com determinadas variáveis do processo;
- Gerar relatórios e planeamento de tarefas;
- Apresentar gráficos de histórico do comportamento das variáveis, que deve ficar guardado, para permitir ao utilizador o estudo do mesmo, definindo posteriormente alguma acção a tomar de maneira a melhorar a produtividade;
- Disparar alarmes que podem ser geridos pelo utilizador. Os alarmes devem ficar registados em *logs*, para o utilizador poder aceder ao seu histórico. É ainda possível haver uma hierarquização de alarmes e distribui-los por utilizador.

Estes sistemas trazem vantagens competitivas pois, com a sua utilização, as empresas podem diminuir a quantidade de trabalhadores nos postos de supervisão das máquinas uma vez que um só trabalhador é capaz de supervisionar várias máquinas de um processo. Estes sistemas oferecem também rapidez na leitura dos instrumentos de campo. Assim, as intervenções necessárias podem ser feitas mais rapidamente o que é um enorme benefício quando se procura o aumento da produtividade.

Actualmente a elaboração de um sistema SCADA não requer qualquer conhecimento de programação uma vez que toda a sua configuração é feita através de ambientes gráficos.

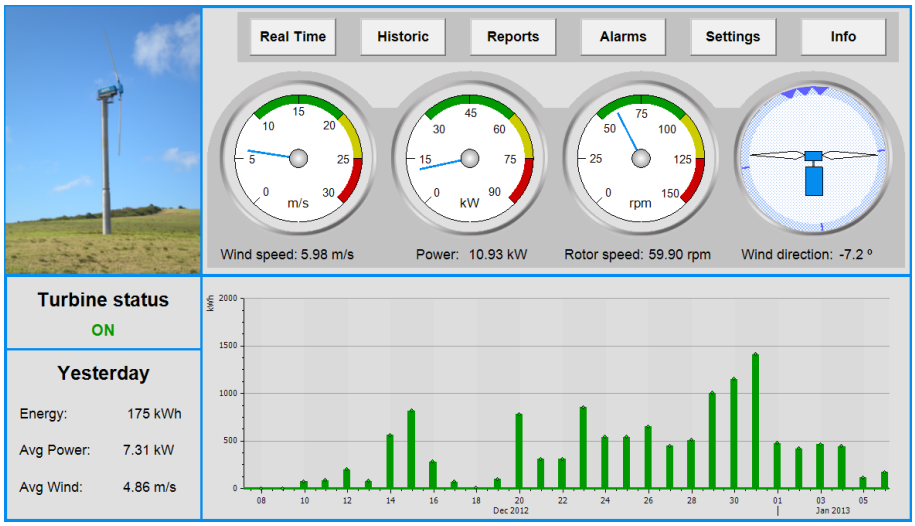

Figura 2 - Interface de um sistema SCADA [8]

2.3 Licenças

Neste subcapítulo explica-se em que consiste uma licença de *software* livre, uma licença de código aberto e uma licença de *software* proprietário, para as quais se dão alguns exemplos.

A licença de *software* livre garante o direito de modificar e redistribuir o conteúdo do *software* sob essa licença. Normalmente estas acções são proíbidas pela lei *copyright* que serve para proteger o autor de documentação ou de *software* de plágio do seu trabalho. Ou seja, o *copyright* não permite que o utilizador copie o código fonte do *software* nem que utilize esse mesmo código para fazer novo código para distribuição (características de uma licença de *software* proprietário). A eliminação destas restrições dá origem às licenças livres [9].

A licença de *software* livre baseia-se em quatro liberdades:

- **Liberdade 0** - A liberdade de executar o programa, para qualquer propósito
- **Liberdade 1** - A liberdade de estudar como o programa funciona e adaptá-lo para as suas necessidades. O acesso ao código-fonte é um pré-requisito para esta liberdade.
- **Liberdade 2** - A liberdade de redistribuir cópias de modo a ajudar o próximo.
- **Liberdade 3** - A liberdade de aperfeiçoar o programa, e distribuir as suas alterações, de maneira a que toda a comunidade beneficie deles. O acesso ao código-fonte é um pré-requisito para esta liberdade [10].

Nem todos os utilizadores e desenvolvedores de *software* livre concordaram com os objectivos do movimento do *software* livre e, em 1998, uma parte da comunidade do *software* livre iniciou uma campanha em nome do "código aberto" dando origem à OSI. Qualquer licença de *software* livre é também uma licença de código aberto, mas o contrário nem sempre se verifica. Os dois termos descrevem quase a mesma categoria de *software*, porém apoiam-se em visões baseadas em valores fundamentalmente diferentes. O código aberto é uma filosofia de desenvolvimento; o *software* livre é um movimento social. Para o movimento do *software* livre, o *software* livre é um imperativo ético, pois apenas este respeita a liberdade dos utilizadores. Em contrapartida, a filosofia do código aberto considera os problemas em termos de como tornar o *software* "melhor", apenas no sentido prático. Esta filosofia diz que o *software* não-livre é uma solução inferior para o problema prático em questão. Para o movimento do *software* livre, contudo, o *software* não-livre é um problema social e a solução é parar de utilizá-lo e migrar para o *software* livre [11].

A definição oficial de "*software* de código aberto" (*open source*) foi indiretamente derivada dos critérios para o *software* livre. Foi criada pela *Open Source Iniciative* (OSI) a partir do texto original da *Debian Free Software Guidelines* (DFSG) e determina que um programa de código aberto deve garantir:

1. **Distribuição livre -** A licença não deve restringir de nenhuma maneira a venda ou distribuição do programa gratuitamente, como componente de outro programa ou não.
2. **Código fonte -** O programa deve incluir o código fonte e deve permitir a sua distribuição também na forma compilada. Se o código fonte não for distribuído com o programa, deve haver algum meio de se obter o mesmo seja via rede ou com custo de reprodução apenas. O código deve ser legível e inteligível para qualquer programador.
3. **Trabalhos Derivados -** A licença deve permitir modificações e trabalhos derivados, e deve permitir que eles sejam distribuídos sob os mesmos termos da licença original.
4. **Integridade do autor do código fonte -** A licença apenas pode restringir o código fonte de ser distribuído numa forma modificada se a licença permitir a distribuição de arquivos *patch* (de actualização) com o código fonte para o propósito de modificar o programa no momento da sua construção. A licença deve permitir explicitamente a distribuição do programa construído a partir do código fonte modificado. Contudo, a licença pode ainda requerer que programas derivados tenham um nome ou número de versão diferentes do programa original.
5. **Não discriminação contra pessoas ou grupos -** A licença não pode ser discriminatória contra qualquer pessoa ou grupo de pessoas.
6. **Não discriminação contra áreas de atuação -** A licença não deve restringir qualquer pessoa de usar o programa num ramo específico de actuação. Por exemplo, não deve proibir que o programa seja usado numa empresa, ou de ser usado para pesquisa genética.
7. **Distribuição da Licença -** Os direitos associados ao programa devem ser aplicáveis para todos aqueles cujo programa é redistribuído, sem a necessidade da execução de uma licença adicional para estas partes.
8. **Licença não específica a um produto -** Os direitos associados ao programa não devem depender que o programa seja parte de uma distribuição específica de programas. Se o programa é extraído desta distribuição e usado ou distribuído dentro dos termos da licença do programa, todas as partes para quem o programa é redistribuído devem ter os mesmos direitos que aqueles que são garantidos em conjunto com a distribuição de programas original.
9. **Licença não restrinja outros programas -** A licença não pode colocar restrições noutros programas que são distribuídos juntamente com o programa licenciado. Isto é, a licença não pode especificar que todos os programas distribuídos no mesmo dispositivo de armazenamento sejam programas de código aberto.
10. **Licença neutra em relação a tecnologia -** Nenhuma cláusula da licença pode estabelecer uma tecnologia individual, estilo ou interface a ser aplicada no programa [9].

2.3.1 Copyleft

Enquanto que o *copyright* é visto pelos mentores originais do *copyleft* como uma maneira de restringir o direito de fazer e distribuir cópias de um determinado trabalho, uma licença de *copyleft* utiliza a lei do *copyright* de forma a garantir que todos os que recebam uma versão da obra possam utilizar, modificar e também distribuir tanto a obra como as suas versões derivadas.

Entende-se, a partir da explicação acima, que o *copyleft* é um mecanismo jurídico para se garantir que os detentores dos direitos de propriedade intelectual possam licenciar a utilização das suas obras além dos limites da lei, ainda que amparados por esta.

A maneira mais simples de tornar um programa livre, é disponibilizá-lo sem copyright. O problema que reside nesta acção é que qualquer pessoa se poderia apoderar do código, podendo depois transformá-lo em *software* proprietário. O *copyleft* impede que o utilizador faça isto, uma vez que diz que qualquer pessoa que distribua *software*, com ou sem modificações, é obrigado a permitir essa mesma liberdade a todos os outros utilizadores. Este mecanismo serve para proteger o *software* livre [12].

A licença GNU GPL é o maior exemplo de uma licença *copyleft*.

2.3.2 Licenças Permissivas

As licenças permissivas impõe poucas restrições às pessoas que obtêm o produto. Nestas licenças não é feita nenhuma restrição ao licenciamento de trabalhos derivados, que podem inclusivamente vir a ser distribuídos sob uma licença fechada. São exemplos deste tipo de licenças as licenças BSD, MIT e Apache. São uma óptima opção para projectos cujo objectivo é atingir o maior número de pessoas, não importando se na forma de *software* livre ou de *software* proprietário.

Alguns programadores argumentam que a utilização desse tipo de licença não incentiva o modelo de *software* livre, pois as empresas aproveitam-se da comunidade para desenvolver *software* que será fechado. Um caso muito conhecido é o do *Kerberos*, desenvolvido no MIT, que posteriormento foi adoptado pela *Microsoft*, que desenvolveu extensões fechadas [13].

2.3.3 Projecto GNU

O GNU é o projecto mais famoso dos que suportam o *software* livre, tendo lançado inúmeras licenças deste tipo de *software*. Assim, devido à sua importância, este subcapítulo pretende explicar em que consiste este projecto e como é que este surgiu.

O sistema operacional GNU é um sistema de *software* livre completo, compatível com o Unix. GNU significa “GNU's Not Unix” (GNU Não é Unix). O anúncio inicial do Projeto GNU foi feito por Richard Stallman em Setembro de 1983. O utilizador pode ou não pagar para obter *software* do projecto GNU. De qualquer forma, uma vez que se possua o *software*, o

utilizador tem que respeitar as quatro liberdades nas quais se baseia um *software* livre, enumeradas no terceiro parágrafo do subcapítulo 2.3.

O Projeto GNU foi concebido em 1983 como uma maneira de trazer de volta o espírito cooperativo que prevalecia na comunidade de computação nos seus primórdios, removendo os obstáculos à cooperação impostos pelos donos de software proprietário. A partir da década de 1980, quase todo o *software* era proprietário o que tornou o Projecto GNU necessário. Para a utilização do computador, é necessário um sistema operativo. Uma vez que não existe nenhum SO livre, é impossível utilizar um computador sem recorrer a um *software* proprietário. Assim, o Projecto GNU decidiu criar um sistema operativo compatível com o *Unix*, porque o seu design geral já era testado e portável, e porque a compatibilidade facilita que utilizadores de *Unix* migrem para o GNU.

Um sistema operativo do tipo *Unix* inclui um *kernel*, compiladores, editores, formatadores de texto, clientes de e-mail, interfaces gráficas, bibliotecas, jogos e muitas outras *features*. Portanto, escrever todo um sistema operativo é um grande trabalho. O desenvolvimento do SO começou em janeiro de 1984 e, por volta de 1990, já se tinham encontrado ou escrito todos os componentes principais, excepto um — o *kernel*. Então o *Linux*, um *kernel* do tipo *Unix*, foi desenvolvido por Linus Torvalds em 1991 e transformado em *software* livre em 1992. Combinar o *Linux* com o quase completo sistema GNU resultou num sistema operativo completo: o sistema GNU/Linux. As estimativas dizem que dezenas de milhões de pessoas utilizam sistemas GNU/Linux.

O Projeto GNU não se limita ao cerne do sistema operativo. Este visa fornecer todo um espectro de *software*, isso inclui aplicações, jogos e outros programas recreativos.

Não existem limites para o que o *software* livre pode alcançar, excepto quando leis, tais como o sistema de patentes, impedem o *software* livre. O derradeiro objetivo do Projecto GNU é prover *software* livre para fazer tudo aquilo que os utilizadores de computadores desejem fazer, tornando assim o *software* proprietário algo do passado [10].

Assim, de forma a proteger os *softwares open source*, o Projecto GNU lançou também diversas licenças que serão abordadas nos próximos subcapítulos. As licenças do Projeto GNU são válidas em todos os países que aceitam o acordo internacional de respeito a patentes e direitos de autor.

2.3.3.1 GNU General Public License (GNU GPL)

A GNU *General Public License* é uma licença livre, copyleft, para *software* e outro tipo de trabalhos.

Em termos gerais, a GPL baseia-se no cumprimento das quatro liberdades que definem a licença livre de *software* (enunciadas no capítulo 2.3). Com a garantia destas liberdades, a GPL permite que os programas sejam distribuídos e reaproveitados, mantendo, porém, os direitos do autor por forma a não permitir que essa informação seja usada de uma maneira

que limite as liberdades originais. A licença não permite, por exemplo, que o código seja apoderado por outra pessoa, ou que sejam impostos sobre ele restrições que impeçam que seja distribuído da mesma maneira que foi adquirido.

A GPL está redigida em inglês e actualmente nenhuma tradução é aceite como válida pela *Free Software Foundation*, com o argumento de que há o risco de introdução de erros de tradução que poderiam deturpar o sentido da licença. Deste modo, qualquer tradução da GPL é não-oficial e meramente informativa, mantendo-se a obrigatoriedade de distribuir o texto oficial em inglês com os programas [9] [15].

2.3.3.2 GNU Lesser General Public License (GNU LGPL)

A GNU *Lesser General Public License*, escrita por Richard Stallman e Eben Moglen em 1991 (e atualizada em 1999), é uma licença de *software* livre aprovada pela FSF e escrita como um meio-termo entre a GPL e licenças mais permissivas, tais como a licença BSD e a licença MIT. Esta licença incorpora os termos e condições da versão 3 da GPL, com a adição de algumas permissões:

0. Additional Definitions
1. Exception to Section 3 of the GNU GPL
2. Conveying modified versions
3. Object Code Incorporating Material from Library Header Files.
4. Combined Works
5. Combined Libraries
6. Revised Versions of the GNU Lesser General Public License

A principal diferença entre a GPL e a LGPL é que esta permite também a associação com programas que não estejam sob as licenças GPL ou LGPL, incluindo *software* proprietário. Outra diferença significativa é que os trabalhos derivados, que não estejam sob a LGPL, devem estar disponíveis em bibliotecas. Ou seja, a LGPL acrescenta restrições ao código fonte desenvolvido, mas não exige que seja aplicada a outros *softwares* que empreguem o seu código, desde que este esteja disponível na forma de uma biblioteca [9] [16] [18].

2.3.3.3 GNU Affero General Public License (GNU AGPL)

A *Affero General Public License* ou *Affero* GPL ou AGPL, como também é informalmente chamada refere-se a duas licenças de *software* distintas. A licença *Affero General Public* - publicada em Março de 2002 pela *Affero* Inc., baseada na licença GPLv2 – e a licença GNU *Affero General Public Licence* – publicada pela FSF em Novembro de 2007, baseada na versão 3 da GPL, a GPLv3.

Ambas as versões da AGPL foram feitas com o intuito de suprir o ASP (*Application Server Provider*) "loophole" presente na licença GPL que consiste na utilização de código fonte aberto de determinado *software* onde apesar de não se fazer a distribuição do mesmo, este é

utilizado para w*eb services*. Na versão GPL não existe protecção copyleft contra este facto. Assim, ambas as versões da AGPL são baseadas na licença GNU GPL à qual se adicionou uma secção adicional que diz que o código de um SaaS (*Software as a service* é uma forma de distribuição e comercialização de *software*) tem que estar disponível para qualquer utilizador desse mesmo serviço.

A FSF recomenda que a GNU AGPLv3 – que foi aprovada como licença *open source* pela OSI em 2008 - seja considerada para qualquer *software* que corre numa rede, tipicamente as aplicações web [17] [18].

2.3.3.4 GNU Free Documentation License (GNU FDL)

A GNU *Free Documentation License* é uma licença para documentos e textos livres que, entre outros, cobre a Wikipedia. Foi publicada pela *Free Software Foundation* e é inspirada na GNU *General Public License*, da mesma entidade. A FDL permite que textos, apresentações e conteúdo de páginas da internet sejam distribuídos e reaproveitados, mantendo, porém, alguns direitos de autor e não permitindo que essa informação seja usada de maneira indevida. A licença não permite, por exemplo, que o texto seja transformado em propriedade de outra pessoa que não o autor, ou que sofra restrições ao ser distribuído da mesma maneira que foi adquirido.

Todos os documentos lançados sob esta licença podem ser utilizados para qualquer propósito caso cumpra determinadas condições:

- Deve atribuir-se o trabalho realizado previamente ao seu autor.
- Todas as alterações ao código devem ser reportadas.
- Todos os trabalhos realizados a partir de determinado código fonte devem estar sob a mesma licença.
- O texto completo da licença e todas as notas *copyright* das versões anteriores devem ser mantidas sem qualquer modificação.
- Medidas técnicas, como a DRM (*Digital Rights Management* - Estas tecnologias são comumente chamadas de tecnologias anti-cópia ou contra a cópia e estão referidas na lei portuguesa - Código de Direito de Autor e Direitos Conexos - como Medidas Tecnológicas ou de Carácter Tecnológico [20]) não devem ser utilizadas para controlar ou obstruir a distribuição ou edição dos documentos [75].

2.3.4 Licença BSD (Berkeley Software Distribution)

A licença BSD foi a primeira licença de *software* livre escrita e é ainda hoje uma das mais utilizadas. Foi criada originalmente pela Universidade da Califórnia em Berkeley para o seu sistema operativo derivado do *Unix* chamado *Berkeley Software Distribution*. A licença BSD é utilizada como modelo por uma ampla gama de *software* licenciado de modo permissivo.

Originalmente a licença BSD possuía uma cláusula que determinava que todo o material de divulgação relacionado com o *software* precisava de conter uma afirmação que dizia "este produto inclui *software* desenvolvido pela Universidade da California, Berkeley e os seus contribuidores". Esta cláusula, que ficou conhecida como "cláusula de propaganda da licença BSD" foi retirada em 1999, originando a "Licença BSD Simplificada".

Através dos termos da licença BSD, o detentor dos direitos de autor permite que outras pessoas usem, modifiquem e distribuam o *software*. A única exigência é que o nome do autor original não seja utilizado em trabalhos derivados sem permissão, visando proteger a sua reputação, dado que o autor pode não ter relação alguma com as modificações realizadas. No caso de redistribuição do código fonte ou binário, modificado ou não, é necessário que seja mencionado o *copyright* original e os termos da licença.

É importante salientar que a exigência de reproduzir a lista de condições para redistribuição e o aviso legal de isenção de garantias não significa que o trabalho a ser redistribuído precise de estar sob a mesma licença [9] [18].

2.3.5 Licença MIT (Massachussets Institute of Technology License)

A licença MIT, criada pelo *Massachussets Institute of Technology*, é também conhecida como Licença X11 ou X, por ter sido redigida para o X Window System, desenvolvido no MIT em 1987.

Esta é também uma licença permissiva e é considerada equivalente à BSD Simplificada. Porém, o seu texto é bem mais explícito ao tratar dos direitos que estão a ser transferidos, afirmando que qualquer pessoa que obtém uma cópia do *software* e dos seus arquivos de documentação associados pode lidar com eles sem restrição, incluindo os direitos a utilizar, copiar, modificar, misturar, publicar, distribuir, sublicenciar e/ou vender cópias do *software* sem qualquer limitação, direitos apenas implícitos na BSD. As condições impostas para tal são apenas manter o aviso de copyright e uma cópia da licença em todas as cópias ou porções substanciais do *software*.

Esta licença é a recomendada pela FSF quando se procura uma licença permissiva, pois é bastante conhecida e, ao contrário da BSD, não possui múltiplas versões com cláusulas que podem gerar dificuldades adicionais, tais como a cláusula de propaganda da BSD que pode gerar incompatibilidades com outras licenças [9] [18].

2.3.6 Licença ASL (Apache Software License)

A licença *Apache* está actualmente na versão 2.0 e é utilizada por um dos projectos mais conhecidos de *software* livre, o servidor web *Apache*.

A *Apache* também é uma licença permissiva e, na versão 1.1, o seu texto era bastante similar ao da BSD, porém dando ênfase à protecção da marca Apache. Havia uma cláusula similar à cláusula de propaganda da BSD, obrigando que a documentação ou o *software*,

quando redistribuídos, incluíssem a frase "este produto inclui *software* desenvolvido pela *Apache Software Foundation* (http://www.apache.org)" e duas cláusulas a proibir a utilização do nome *Apache* sem permissão escrita prévia, tanto para publicar trabalhos derivados como também para a utilização como parte do nome do produto. Em 2004, a licença foi totalmente reescrita e o seu texto ficou bem mais longo e complexo, detalhando melhor os direitos concedidos.

Quanto aos direitos de autor, é dada uma licença irrevogável para reproduzir, preparar trabalhos derivados, mostrar publicamente, sublicenciar e distribuir o trabalho e derivados deste, na forma de fonte ou objecto, sujeitos aos termos e condições de licença. Existem algumas condições para a redistribuição do trabalho e derivados do mesmo:

- Incluir uma cópia da licença;
- Incluir avisos em todos os arquivos modificados informando sobre a alteração
- Manter na fonte de trabalhos derivados todos os avisos de direitos de autor, patentes e marcas registadas que são pertinentes;
- Se o trabalho incluir um arquivo de texto chamado "NOTICE", então qualquer trabalho derivado distribuído deve incluir os avisos pertinentes contidos nesse arquivo da forma como está detalhado na licença.

A cláusula sobre redistribuição termina informando explicitamente que é permitido licenciar sob outros termos qualquer modificação ou trabalho derivado, desde que a utilização, reprodução e distribuição do trabalho que foi obtido pela licença Apache esteja de acordo com os seus termos.

O facto da licença *Apache* deixar explícito na cláusula sobre redistribuições que é permitida a utilização de outra licença, é visto como uma vantagem sobre as licenças BSD ou MIT quanto à clareza da sua característica permissiva. O principal problema da *Apache* é o facto da sua compatibilidade com a GPL 2.0 ser discutível [9] [18].

2.4 Protocolo Modbus

Neste subcapítulo aborda-se o protocolo *Modbus*, mais especificamente a sua implementação TCP/IP. Este protocolo foi utilizado para a comunicação de ambos os programas (automatização e SCADA) realizados no âmbito desta dissertação com o simulador da LPF.

2.4.1 Visão Geral

O *Modbus* é um protocolo de comunicação aberto, introduzido no mercado em 1979 pela Modicon. A sua simplicidade de implementação tornou-o muito popular no seu principal domínio de aplicação, a Automação Industrial. Não sendo objecto de normalização, viu na Modbus-IDA (uma organização sem fins lucrativos composta por fabricantes e aberta a integradores e utilizadores) [21] a principal responsável pela divulgação e evolução do

protocolo, tendo publicado um conjunto de documentos de referência para a implementação, bem como gerido aplicações que visam testar a conformidade dos dispositivos que queiram comunicar usando este protocolo [22].

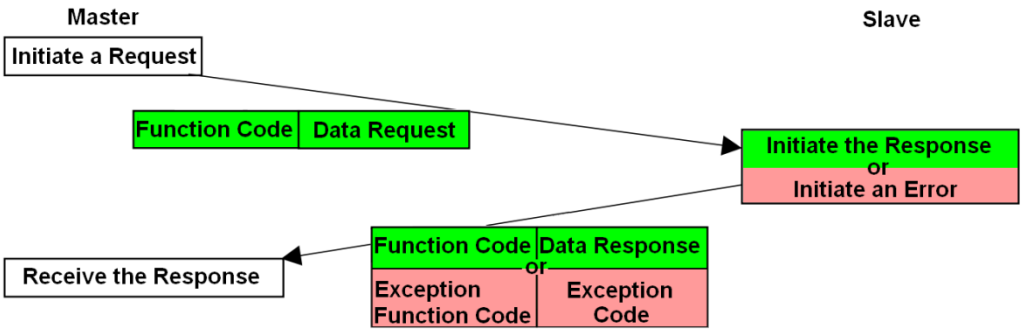

Figura 3 - Modbus request and reponse [19]

O *Modbus* é um protocolo que se situa na camada de aplicação do modelo OSI para comunicação entre dispositivos em rede, essencialmente para troca de dados no campo da automação. Nesse nível, o protocolo segue o paradigma cliente-servidor, e diz-se *stateless*, ou seja, uma dada troca de informação é totalmente independente do histórico de trocas de informação precedentes ou eventualmente em curso. Os referidos pedidos e respostas baseam-se em tramas simples, designadas por Protocol Data Unit (PDU), independentes das camadas inferiores. A especificação [21] define três tipos de PDU's (ver Figura 3):

- Request PDU:
    1. um código que indica uma função (1 byte)
    2. dados correspondentes à função (tamanho variável)

- Response PDU
    1. o código da função correspondente ao pedido (1 byte)
    2. dados correspondentes à resposta (tamanho variável)

- Exception Response PDU:
    1. o código da função correspondente ao pedido + 0x80 (128) (1 byte)
    2. um código identificador do tipo de excepção (1 byte)

2.4.2 Serviços

A especificação do protocolo define um conjunto de funções, cada uma das quais com o seu código único. Estes códigos encontram-se no intervalo [1-127], sendo que o intervalo [129-255] está reservado para os códigos de excepção na resposta, como foi referido anteriormente [21]. São definidas também três categorias de códigos de funções:

- •*Public* – São garantidamente únicos, e estão associados a funções bem definidas e documentadas. São validadas pela organização *Modbus-IDA* e existem testes de conformidade para as mesmas. Como são funções standard, devem estar definidas em todos os equipamentos que utilizem o protocolo Modbus.

Tabela 1 - Funções Modbus [21]

Código da função Modbus	Nome	Função
01	Read coil status	Reads the bit data (N bits)
02	Read input discrete value	Reads the bit data
03	Read multiple registers	Reads the integer type/character type/status word/floating point data (N words)
04	Read input registers	Reads the integer type/status word/floating point type data
05	Write single coil	Writes the bit data
06	Write single register	Writes the integer type/character type/status word/floating point type data (1 word)
15	Write multiple coils	Writes the bit data (N bits)
16	Write multiple registers	Writes the integer type/character type/status word/floating point data (N words)

- *User Defined* - A especificação [21] define os intervalos [65-72] e [100-110] como códigos de funções para implementação livre por parte dos utilizadores. Os respectivos códigos não são garantidamente únicos, pois dependem dessas mesmas implementações. Estas funções só estarão disponíveis nos equipamentos onde o utilizador as implementar.
- *Reserved* - Estes códigos são usados por alguns fabricantes para soluções proprietárias e não estão disponíveis para uso público. Não são se quer discutidos na especificação, sendo o leitor remetido para o anexo A do documento [21] para detalhes sobre os códigos reservados.

Uma função encontra-se documentada com:

1. a descrição da função, isto é, o seu propósito, os seus parâmetros, e valores de retorno (incluindo eventuais códigos de erro nas excepções).
2. o código da função que lhe está associado.
3. o formato de cada uma das três PDU's - *Request, Response e Exception Response*

2.4.3 Modelo de dados

As funções públicas básicas foram desenvolvidas para troca de dados na área da automação. Na tabela 2 apresentam-se os tipos de dados básicos que o documento [21] especifica. Na implementação de um servidor deve documentar-se a organização dos dados.

Tabela 2 - Tipos de dados básicos usados no protocolo Modbus [21]

Nome	Tipo	Acesso
Discrete Input	1 bit	Somente leitura
Discrete Output	1 bit	Leitura/escrita
Input Registers	Palavra 16 bits	Somente leitura
Output Registers	Palavra 16 bits	Leitura/escrita

Salienta-se o facto de que a nomenclatura usada para o mesmo tipo de dados varia frequentemente na literatura, incluindo a própria especificação (por exemplo o tipo Discrete Output é referido como Coil, ou o tipo Output Registers como Holding Registers), facto que se propaga também para os nomes usados nas funções de acesso a estes tipos de dados [12].

2.4.4 Implementação TCP/IP

O protocolo Modbus conhece diversas implementações, sendo que as mais populares trabalham sobre TCP/IP e sobre transmissões série assíncronas (onde os meios físicos mais comuns são RS-232 e RS-485). Pode-se encontrar no documento [23] a especificação da implementação TCP/IP, para além de descrições funcionais de um cliente, servidor e *gateway* (dispositivo que garante a interface entre a rede IP e os barramentos série).

O protocolo define uma trama (PDU) cujo formato é independente da implementação. No Modbus/TCP é adicionado um cabeçalho específico, formando assim uma trama própria da implementação, denominada de Application Data Unit (ADU), como mostra a Fgura 4. Esse cabeçalho, denominado de MBAP na especificação, tem 7 bytes de comprimento, e é composto por:

1. transaction identifier (2 bytes) - usado para identificação unívoca das mensagens (pedidos e respostas), pois é copiado pelo servidor na resposta a um determinado pedido;

2. protocol identifier (2 bytes) - tem o valor zero por defeito para o *Modbus*, existindo essencialmente na expectativa de expansões futuras;
3. lenght (2 bytes) - contém o número de bytes que se seguem na trama, por forma a ajudar na detecção dos limites da mesma;
4. unitidentifier (1byte) - usado para identificação de um dispositivo remoto localizado numa rede distinta da rede TCP/IP, sendo o elemento que é usado pelas gateways para direccionar um pedido que lhe seja endereçado [23].

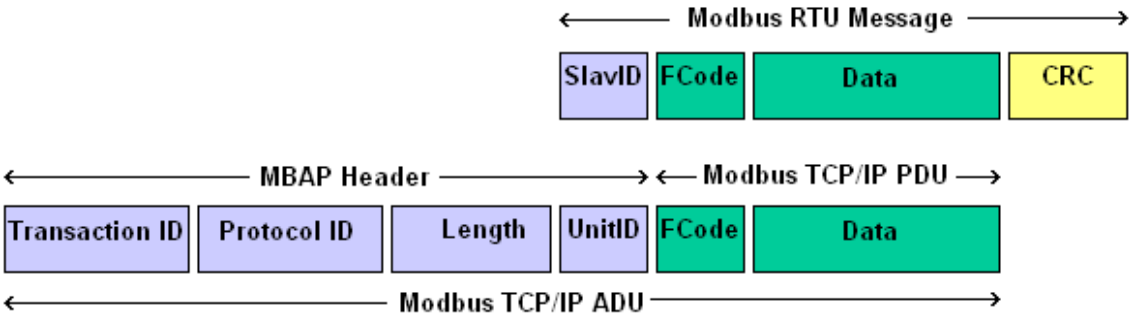

Figura 4 - Application Data Unit do protocolo Modbus TCP/IP [24]

As configurações e topologias de uma rede Modbus TCP/IP não se encontram definidas na especificação, sendo apresentados apenas exemplos ilustrativos. É perfeitamente viável construir redes com mais do que um cliente, ou mesmo ter dispositivos que funcionem simultaneamente como cliente e servidor. Esta particularidade é vista como uma grande vantagem da implementação TCP/IP do Modbus face às restantes [22].

Capítulo 3

Estado da Arte

O Estado da Arte destina-se a documentar o que está a ser feito actualmente no campo em estudo e é fundamental para explicar os acréscimos que a dissertação fará ao estado de conhecimento atual.

Assim, de maneira a documentar o que já existe a nível de ferramentas *open source* para a automatação industrial, procedeu-se à pesquisa de ferramentas tanto para a automatização de processos como para desenvolvimento de interfaces SCADA.

3.1 Ferramentas de automatização de processos

Quanto à automatização de processos, procuraram-se *Integrated* D*evelopment Environments* (IDE) para a programação seguindo preferencialmente a norma *international industrial standard IEC 61131-3* cuja utilização não dependesse da existência de um PLC.

Encontraram-se inúmeras soluções como o OpenPCS Automation Suite [25] ou o CODESYS (COntroller DEvelopment SYStem) [26], que são IDE's para o desenvolvimento de *software* para programação de aplicações de controlo de acordo com a norma *international industrial standard IEC 61131-3,* que têm a capacidade de aliar as 5 linguagens da norma e ainda a linguagem CFC (*Continuous Function Chart)*.

O CODESYS tem compiladores internos que transformam o código da aplicação em código de máquina (código binário) que é descarregado num controlador. Desta forma, esta ferramenta não necessita de um PLC, delegando as suas tarefas para um microcontrolador. A maioria das famílias importantes de CPU de 16-bit e 32-bit são suportadas, tais como *C166, TriCore, 80x86, ARM/Cortex, Power Architecture, SH, MIPS, BlackFin*, por exemplo.

Apesar do potencial destas ferramentas, estas não pertencem à categoria *free software*. Deve entender-se *free* como livre (dá liberdades ao utilizador) e não como grátis (sem qualquer custo para o utilizador).

Após uma pesquisa aprofundada, concluiu-se que o único IDE *open source* em conformidade com a norma IEC 61131-3 é o *Beremiz*.

Beremiz

O Beremiz é um IDE de código fonte aberto, multiplataforma, para o desenvolvimento de programas de automação em conformidade com o disposto na norma IEC 61131-3, disponibilizada ao público livremente sob a licença de software GNU GPLv2.1 ou posterior s Permite a substituição do PLC por qualquer processador. Foi desenvolvido de forma modular e baseia-se nos sub-projectos seguintes:

- PLCOpen Editor - Editor dos programas IEC 61131-3
- MATIEC - Compilador IEC 61131-3 -> ANSI-C
- CanFestival - CANOpen Framework para interface com I/O físicos
- SVGUI - Ferramenta para integração de HMIs

O compilador MATIEC converte o projecto desenvolvido nas linguagens de programação definidas na norma IEC61131-3 em código C equivalente. Funciona em quatro etapas: *lexical analyzer*, *syntax parser*, *semantics analyzer* e *code generator* [27] [28].

MBLogic

Encontrou-se ainda um projecto com algum ineteresse mas se considerará a sua utilização para a implementação desta dissertação uma vez que a linguagem soft logic não é feita seguindo a norma IEC 61131-3.

O projecto *MBLogic* está alojado no portal *http://sourceforge.net/* que é um repositório de código fonte, que actua como um centro para os programadores *developers* gerirem os seus projetos colaborativamente. O *MBLogic* pretende ser uma plataforma completa de automação. Contém um conjunto de pacotes de *software* utilizados para a comunicação e controlo industrial de processos e maquinaria. Este conjunto de pacotes pode ser utilizado individualmente ou em conjunto, conforme o desejado pelo utilizador. Os pacotes que este projecto contém são os seguintes:

- *MBLogic Soft Logic*: este pacote é uma plataforma completa e independente para o controlo de plataformas industriais. Incorpora *soft logic*, *web based* HMI, campo para conectixão IO e ajuda on-line.
- *HMIServer*: pacote independente que fornece ao utilizador um sistema HMI para a monitorização e controlo de dispositivos industriais como o PLC, por exemplo.
- *HMIBuilder*: pacote que proporciona a criação automática de páginas web HMI sem a necessidade de programação. Os *screens* são montados usando *widgets* gráficos fornecidos, utilizando o método *drag and drop*, e configurados através de menus onde se selecionam as opções desejadas.

- *MBLogic Tools* é o pacote que fornece ferramentas para teste e solução de problemas das aplicações. Para além destas funcionalidades, fornece também servidores *gateway* que permitem que os clientes comuniquem uns cons os outros. Algumas das ferramentas existentes neste pacote são: *MBAsyncServer* (servidor proxy *Modbus*/TCP), *MBProbe* (ferramenta cliente web based para a leitura e escrita de um servidor Modbus/TCP para teste e troubleshooting, manualmente), e *MBPoll* (linha de comandos para testar servidores Modbus/TCP).
- *MBLogic Libraries* é um pacote para o desenvolvimento de aplicações em Python. Contém um driver de cliente *Modbus*/TCP e um sistema *soft logic* que podem ser incluídos como bibliotecas nos programas a serem desenvolvidos pelo utilizador. Estas bibliotecas são utilizadas tipicamente para a produção de sistemas de teste. [29]

Existem ainda projectos *open source* que utilizam a programação em *Ladder Logic* (a linguagem LD especificada na norma IEC61131-3 é um exemple de *ladder logic*) como o **Classic Ladder** [30], o **SoapBox Snap** [31] ou o **LDMicro** [32].

3.2 Projectos "Open Hardware"

Existem também diversos projectos *open hardware* cujo objectivo é o de substituir um PLC, utilizando um processador para o mesmo propósito.

O OLinuxINO, visível na Figura 5, é um computador *single-board* da classe industrial com *software* e *hardware open source* com GPIOs (**General Purpose Input/Output** - são basicamente portas programáveis de entrada e saída de dados e são utilizadas para prover uma interface entre os periféricos e os microcontroladores/microprocessadores) capazes de operar de -25ºC até 85ºC [33].

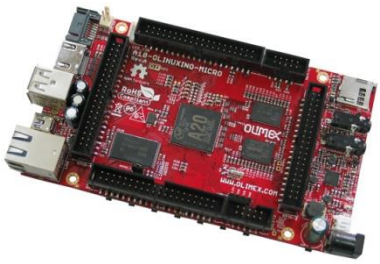

Figura 5 – OLinuxINO [34]

Para o Arduino (fig. 6) e para o Raspberry PI, que são computadores com características parecidas com o OLinuxINO, existem inúmeros projectos para os fazer desempenhar o papel de um PLC, substituindo-o.

Figura 6 – Arduino [35]

Figura 7 - Raspberry PI [36]

Este tipo de processadores, para funcionarem de acordo com a norma IEC 61131-3, têm obviamente que estar associados a uma ferramenta de software capaz de fazer um programa de controlo em conformidade com a mesma(ex: Beremiz - *open software*).

Como exemplo de uma solução *open hardware* que utiliza um micro computador como os citados nos parágrafos anteriores, temos o *OpenPLCLib* (Figura 8). Este é uma alternativa para a automação industrial ou domótica. Utiliza a família de chips *ATMega* (a mesma que é utilizada no Arduíno) como processador, o que faz com que esta solução seja compatível com o Arduíno. Os principais detalhes deste projecto são:

- Entradas digitais de 24V protegidas (8 entradas por cada módulo).
- Saídas digitais de 24 V protegidas (8 entradas por cada módulo).
- Comunicação USB (usada para programar).
- RS-485 bus para comunicação entre módulos.
- IDE open source para programação em C++.
- IDE open source para programação em *Ladder* (O nome *ladder* - escada em inglês - provém do facto que a disposição dos contactos e bobinas ser realizada, de maneira geral, na horizontal, lembrando o formato de uma escada.)
- *Ethernet* integrada.
- 16MHz ATMega2560.

A programação em *Ladder* é baseada no projecto LDMicro, de Jonathan Westhues. Este *software* é capaz de gerar código C a partir de um diagrama *ladder*. Para além deste compilador AVR, o OPenPLCLib contém ainda uma interface de interação com o utilizador (feita em C#) onde este pode compilar o seu diagrama ladder para código C em apenas um clique [37].

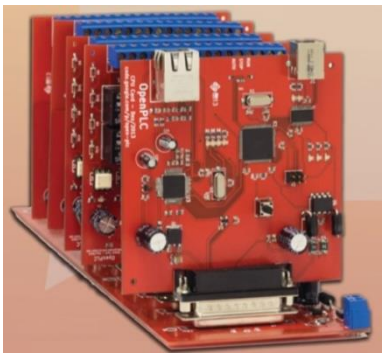

Figura 8 – OpenPLCLib [37]

3.3 Ferramentas para desenvolvimento de interfaces SCADA

Relativamente às ferramentas existentes para o desenvolvimento de interfaces SCADA (*Supervisory Control And Data Acquisition System*), encontraram-se inúmeras soluções *open source*, que serão apresentadas de seguida.

OpenSCADA

O OpenSCADA é uma implementação *open source* de um sistema SCADA. É independente da plataforma utilizada e é baseado no design de sistemas mordernos que proporcionam flexibilidade e segurança ao mesmo tempo. As suas principais propriedades são a modularidade, escalabilidade, abertura (tem licença GPL) e ser multiplataforma.

O *Linux* é o sistema operativo base para o desenvolvimento e utilização desta ferramenta. Para além de ser compatível com o standard POSIX, o *Linux* é ainda uma óptima solução em questões de segurança, disponibilidade, popularidade e flexibilidade/escalabilidade. Como o OpenSCADA foi desenvolvido no standard POSIX-OS, a sua adaptação a outro sistema operativo não é um problema.

A arquitectura deste sistema consiste numa organização em subsistemas, cada um com as suas funções:

- Segurança – contém a lista de utilizadores e grupos de utilizadores. Verifica quem é que pode aceder a cada elemento do sistema.
- DB – acesso à base de dados
- Transporte – providencia a comunicação com o ambiente através de diferentes interfaces de comunicação
- Protocolo de Transporte – está ligado ao subsistema de transporte e dá suporte a vários relatórios de troca com sistemas externos
- DAQ – fornece dados das fontes exteriores como controladores, medidores, etc.

- Arquivo – contém arquivos de dois tipos: arquivos de mensagens e arquivos de valores. O modo de arquivação é definido pelo algoritmo incorporado neste módulo.
- User Interface – contem as funções de UI (User Interfaces)
- Controlo – faz o controlo de todos os módulos
- Módulo Especial – contém funções extra que não existem nos outros subsistema

O *kernel* modular do sistema *OpenSCADA* é realizado sob a forma de bibliotecas estáticas e partilhadas. Permite a construção de funções já existentes no sistema, e permite também criar novos programas. Os módulos do sistema *OpenSCADA* estão alojados em bibliotecas dinâmicas [38].

Este software moveu-se recentemente para a *eclipse*, dando origem ao **Eclipse SCADA**, projecto que ainda não se encontra completamente concluído.

Para além do *OpenSCADA*, encontraram-se ainda diversos *softwares open source* com as mesmas características (sistemas de supervisão completos), a saber:

Mango

É um software M2M (*Machine-to-Machine*) de código aberto. Este software é *browser-based* que permite aos utilizadores o acesso e controlo de sensores, dispositivos electrónicos e máquinas através de diversos protocolos simultaneamente. O *Mango* providencia uma interface na qual é possível criar e configurar diversos tipos de "dispositivos" para monitorização de informação, fornecendo *downstream management*, alertas, *data logging* e automatização [39].

ScadaBR

É um sistema de supervisão completo que conta já com 70000 downloads efectuados e 5 anos de existência. Como o próprio nome indica, é um software brasileiro [40]. Surgiu a partir do Mango M2M. O funcionamento desta ferramenta está detalhado no capítulo 4.2.3.

Proview Open Source Process Control

Originalmente desenvolvido na Suécia pela *Mandator* e pela SSAB *Oxelösund* como um sistema de controlo de processos baseado em computadores standard, o sistema tornou-se num sistema completo, integrado e uma solução barata que corre em PCs standard que tenham *Linux* como sistema operativo. O *software Proview* é *open source* e a licença é GNU/GPL [41].

Szarp

É um projecto desenvolvido por polacos que consiste num sistema SCADA Open Source completo para Linux. Os programas client estão disponíveis tanto para Linux como para Windows [42].

PVBrowser

É uma aplicação *framework* alemã que permite ao utilizador a criação de aplicações SCADA. Providencia ao utilizador um browser especializado para o client (pvbrowser) e um IDE (pvdevelop) para a criação de servidores (pvserver) que implementam a visualização a ser apresentada ao cliente. Funciona com o princípio cliente/servidor e está sob a licença LGPL [43].

Likindoy

O *Likindoy* (desenvolvido em Espanha) é um SCADA feito com tecnologias de código aberto. Este programa foi projectado para análise industrial e para sistemas remotos e foi projetado de forma modular para permitir que outros programas possam ajudar este projecto *open source*. O *Likindoy* está dividido em várias fases de operação mas internamente é dividido em três módulos: o módulo de base que contém as bibliotecas que são utilizadas pelos outros módulos, o módulo de gestão de histórico (Likindoy-HTR) e modo módulo de gestão de RTUs (Likindoy-RTU).

O módulo para a gestão de recursos históricos (HTR) tem 4 níveis de processamento de dados:

1. Colecta de dados: o programa usa diferentes módulos para obter informações a partir de fontes externas. (FTP, SFTP, socket, MODBUS, WEB, UDP).
2. Colocação dos dados: os dados descarregados e transferidos para um banco de dados SQL de forma homogénea.
3. Geração de gráficos : utilizam-se as informações armazenadas na base de dados para gerar gráficos.
4. Apresentação de dados : envio dos gráficos, geralmente por e-mail, sendo também suportados os protocolos FTP, SFTP e socket (os utilizadores podem criar novos módulos para enviar dados através de seus próprios protocolos).

O módulo RTU do *Likindoy* é capaz de recolher dados directamente do hardware industrial através do protocolo de comunicação *Modbus* TCP/IP o que lhe permite comunicar com praticamente qualquer PLC industrial. Este módulo pode gravar os dados de tráfego do

sistema de arquivos de rede obtidos, pode colectar arquivos de dados usando o protocolo FTP e SFTP ou diretamente de um servidor da Web ou servidor MODBUS usando UDP.

O módulo HMI é o módulo encarregado de exibir informações em tempo real [44].

PascalSCADA

É um *framework* para Delphi/Lazarus cuja principal característica é o rápido desenvolvimento de aplicações HMI/SCADA. Este *software* funciona em *Linux*, *Windows* (ambos em 32 e 64 bits) e em *FreeBSD* (32 bits). Com o *PascalSCADA* é possível comunicar com PLCs, criar interfaces, registar variáveis e alarmes do processo e controlar utilizadores da aplicação [45].

Parijat HMI/SCADA Open-source Development System

Este produto foi desenvolvido em *Microsoft Visual Basic.Net* para ambiente *Windows*. Permite a qualquer utilizador construir rapidamente interfaces *Human-Machine* para PLC's, CDS, *Flow Computers* ou *Chromatographs* [46]. É uma solução *open source*!

SCADA Open Source (SOS)

Este *software* foi desenvolvido para permitir o desenvolvimento de SCADA's completos e com performance elevada. Permite ao utilizador executar rotinas *Database Human Machine Interface* (HMI) [47].

Para além dos *softwares* enumerados no parágrafo anterior, encontraram-se ainda alguns projectos com interesse de ser referidos, como é o caso do **VISUAL** [48], do **OpenWebSCADA** [49], do **FreeSCADA** [50] e do **Mellisa** [51]. Estes projectos são softwares HMI/SCADA *open source* que ainda não estão completamente desenvolvidos e/ou testados, não oferecendo ainda a confiabilidade necessária ao utilizador.

Capítulo 4

Ferramentas Open Source Avaliadas

Depois da pesquisa efectuada na procura de *softwares* existentes, passou-se à selecção de quais as ferramentas com interesse de avaliar. Este capítulo dá uma visão mais aprofundada sobre as ferramentas escolhidas.

4.1 Ferramentas de automatização de processos

Para a automatização de processos, apenas se explorou o programa Beremiz, que foi o único programa *open source* para a automatização de processos capaz de programar em conformidade com a norma IEC 61131-3 que se encontrou. Para este programa, realizaram-se testes exaustivos na procura de erros e fez-se um relatório de *bugs* que reporta os erros encontrados.

Um tutorial para a criação de um pequeno projecto nesta ferramenta está anexado a esta dissertação (Anexo A). Achou-se pertinente a elaboração deste tutorial uma vez que não há nenhuma documentação de ajuda ao utilizador para a utilização deste *software*.

4.1.1 Beremiz

4.1.1.1 Visão Geral

O Beremiz é um ambiente de desenvolvimento integrado para o desenvolvimento de programas de automação em conformidade com o disposto na norma IEC 61131-3 e está disponível sob a licença de *software* GNU GPLv2.1, o que significa que este é um IDE livre e de código aberto. As principais motivações para o desenvolvimento de uma ferramenta com estas características foram definidas pelos autores em [27]. Neste artigo, os autores referem que a diversidade de linguagens de programação utilizadas pelos diferentes fabricantes de PLC's faz com que a aprendizagem feita pelo programador, quando se muda de marca de PLC, seja difícil e demore algum tempo. Assim, apesar da normalização sobre a programação de PLC's, há dificuldades na portabilidade dos programas desenvolvidos para os mesmos uma vez que normalmente a programação de PLC's varia de fabricante para fabricante. Para além

disso, o processo de aprendizagem da norma IEC 61131-3 envolve normalmente a aquisição de licenças dispendiosas, limitando ou até impossibilitando a utilização de um *software*, pelos estudantes, nos seus próprios recursos informáticos ao longo do período de ensino não assistido.

O Beremiz foi desenvolvido de forma modular e é composto por vários subprojectos:

- *PLC Open Editor*- Editor dos programas IEC 61131-3
- *MATIEC*- Compilador IEC 61131-3 -> ANSI-C
- *CanFestival*- CANOpen Framework para interface com I/O físicos
- *SVGUI*- Ferramenta para integração de HMIs

Nas secções seguintes focar-se-ão os módulos *PLC Open Editor* e *MATIEC*. Forcar-se-ão ainda de uma forma geral os *Plugins* disponíveis para a ferramenta *Beremiz*, como o *SVGUI* e o *CanFestival*.

4.1.1.2 PLC Open Editor

O PLC Open Editor está programado em Python [52] (*Python* é uma linguagem de programação de alto nível, interpretada, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte [53]), usando o módulo adaptador *WxPython* [54] para a biblioteca *WxWidgets* [55]. Esta ferramenta está fortemente ligada às especificações PLCOpen [6].

É nesta ferramenta que o utilizador do *Beremiz* escreve os seus POU's. A interface confere ao utilizador uma perspectiva global do projecto, apresentando cinco áreas (barra de ferramentas na zona superior, *console* na zona inferior, *debugger*/*library* do lado direito, gestão do projecto do lado esquerdo e escrita/edição de POU's na zona central), visíveis na Figura 9, que serão descritas nesta secção.

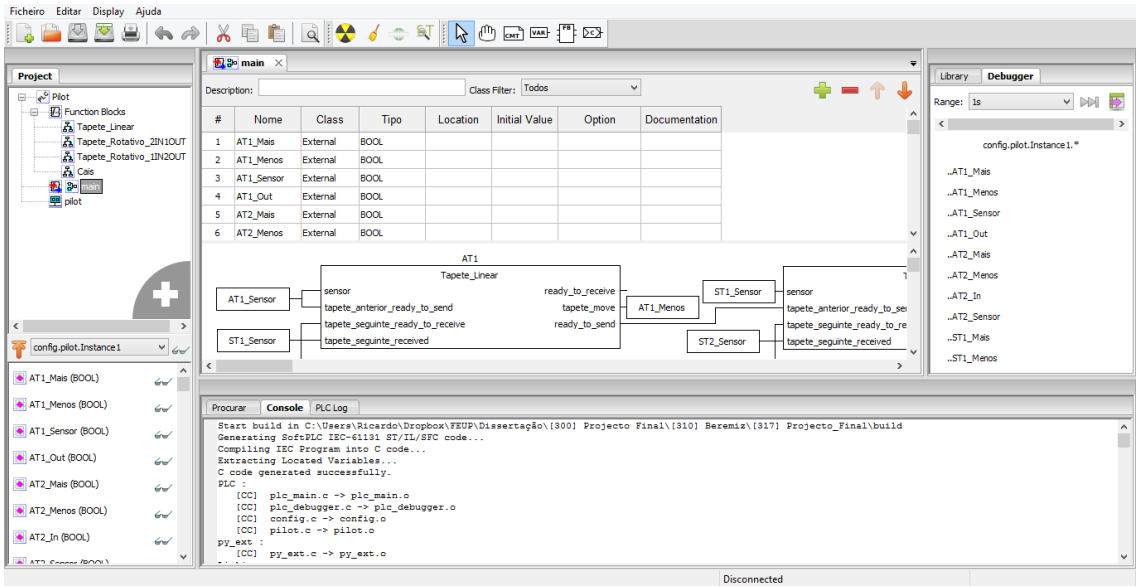

Figura 9 - Interface Beremiz

Na divisão da esquerda, o utilizador faz a gestão do seu projecto. Pode adicionar e remover POU's ou *plugins*, como o *CanOpen*, o *Modbus* ou o *SVGUI*.

A divisão inferior contém a *console* que apresenta informação relativa ao resultado das acções tomadas pelo utilizador.

A escrita das instruções que cada POU deve executar é feita na divisão central, sendo que esta se torna um editor especializado para qualquer uma das cinco linguagens, em função da linguagem definida para o POU que o utilizador esteja a editar no momento. Do ponto de vista gráfico, a zona central apresenta duas divisões: uma para a escrita e edição dos POUs e outra com a lista das variáveis associadas a esses POUs, como mostra a Figura 10.

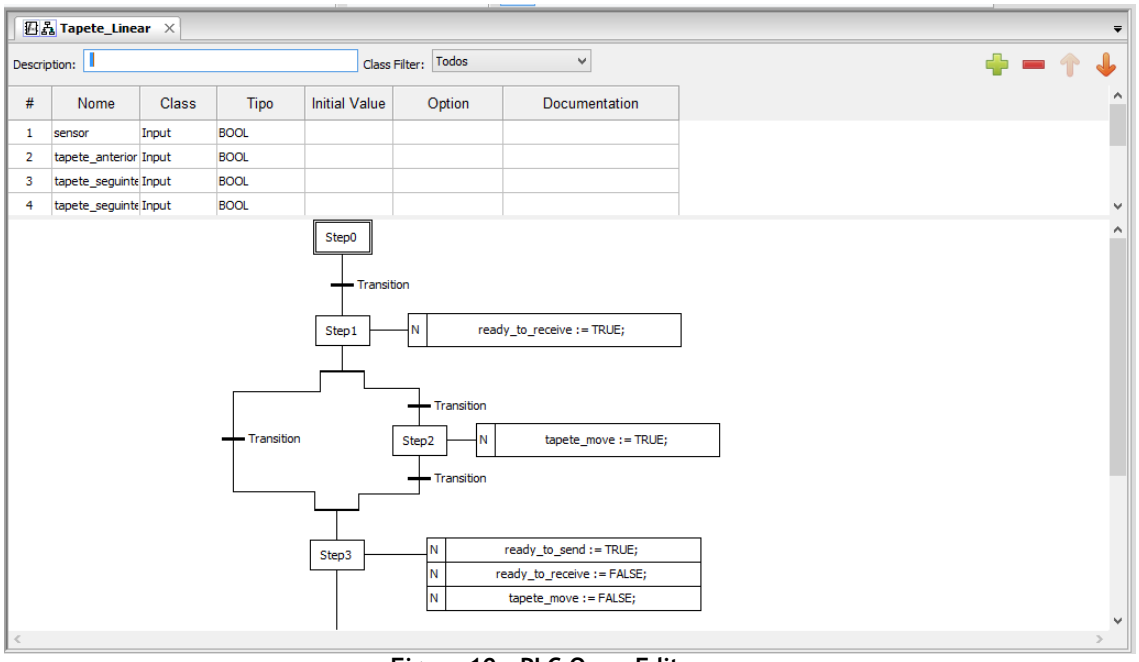

Figura 10 - PLC Open Editor

É possível adicionar e eliminar variáveis dessa lista, bem como editar os seus campos: nome, âmbito, tipo, localização, valor inicial, retenção, e se o identificador deve ser declarado como constante ou não.

É também nesta ferramenta que as configurações e tarefas são geridas, em conformidade com o estabelecido na norma IEC61131-3, tal como ilustrado na Figura 11.

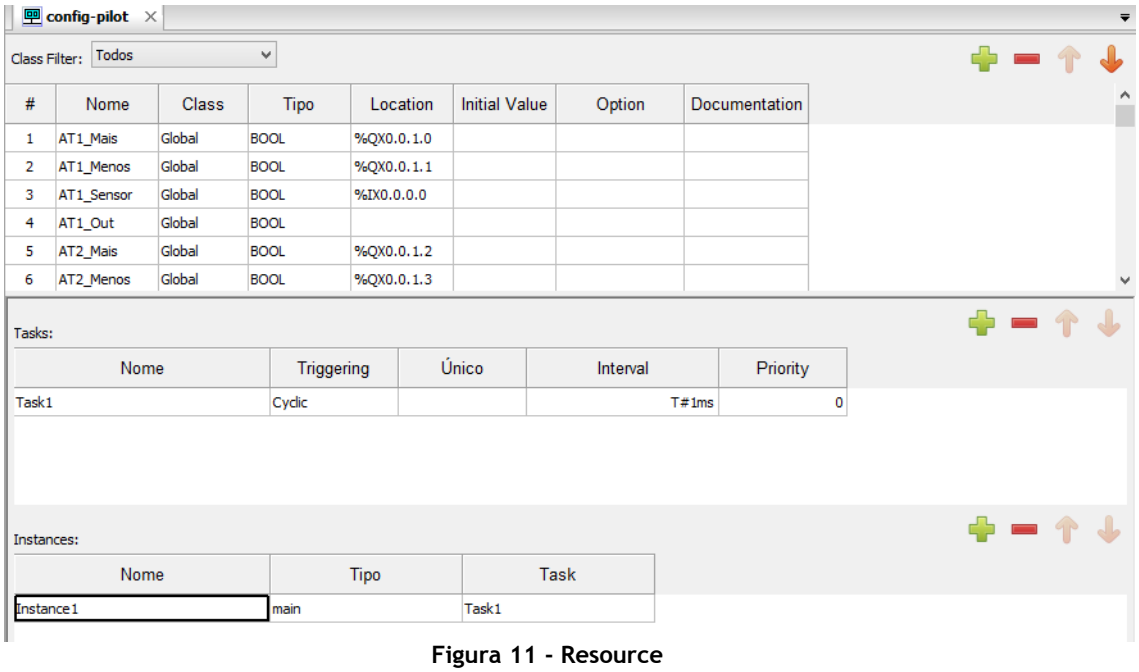

Figura 11 - Resource

Dependendo da linguagem gráfica em uso (LD, FBD ou SFC), aparecem na barra de ferramentas situada no topo, os elementos que a linguagem em uso comporta que podem assim ser adicionados ao POU em questão. Esta barra é dinâmica, ou seja, as opções disponíveis variam com o estado do projecto. Inclui também opções como criar um novo projecto, gravá-lo, compilar o projecto, ver o código IEC 61131-3 gerado e transferir para o SoftPLC, por exemplo.

Na divisão à direita, encontra-se o *debugger*, onde se pode verificar o valor das variáveis em tempo real. As variáveis que se pretendem monitorizar são seleccionados pelo utilizador da listagem de variáveis que se encontra na zona inferior da divisão da esquerda (lista de variáveis declaradas no POU que esteja seleccionado pelo utilizador), visível na Figura 9. Para além do *debugger*, existe também uma outra árvore de selecção na divisão à direita, denominada por *library*, com um conjunto de *functions* e *function blocks*, agrupados por âmbito de utilização, que incluem para além dos POU's da biblioteca padrão, os POU's definidos pelo utilizador. Com a selecção de uma função dessa biblioteca, pode arrastar-se a mesma para a divisão central, o que é especialmente útil nas linguagens gráficas, pois é de imediato apresentado um bloco com a respectiva interface. No caso de ser adicionado um *function block*, é requerido um nome para a instância desse FB, que é automaticamente adicionado à lista de variáveis do POU em edição.

Os projectos são guardados em formato XML, cujo modelo de dados segue a estrutura definida no *XML Official Schema* do comité técnico TC6 da organização PLCOpen [56]. Essa estrutura oficial, que na prática se traduz num ficheiro *.xsd [57], é usada na criação de um projecto e estabelece todas as relações entre os objectos que compõem o mesmo. Essa valência permite que o *PLC Open Editor* seja usado para validar se um determinado projecto nesse formato segue a estrutura base referida. Esta organização está patente na Figura 12. O

PLC Open Editor dispõe ainda de um módulo responsável pela conversão das componentes do projecto que incluam o uso de linguagens gráficas em equivalentes textuais. Em concreto, as linguagens FBD e LD são convertidas em ST equivalente, enquanto que os elementos dos SFC's têm na norma uma especificação textual própria, que é usada para este efeito [58].

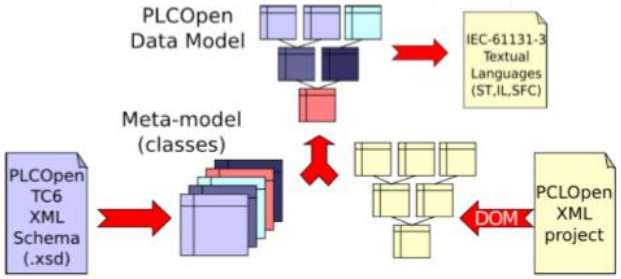

Figura 12 - Herança do modelo de dados TC6-XML Schema e posterior conversão textual [59]

4.1.1.3 Compilador IEC 61131-3

O resultado da conversão textual de um projecto IEC 61131-3, referido no ultimo parágrafo da secção anterior será posteriormente convertido em código C pelo compilador *MATIEC*, cuja organização está presente na Figura 13. O compilador funciona em quatro etapas: *lexical analyzer*, *syntax parser*, *semantics analyzer* e *code generator* que são explicadas detalhadamente em [27].

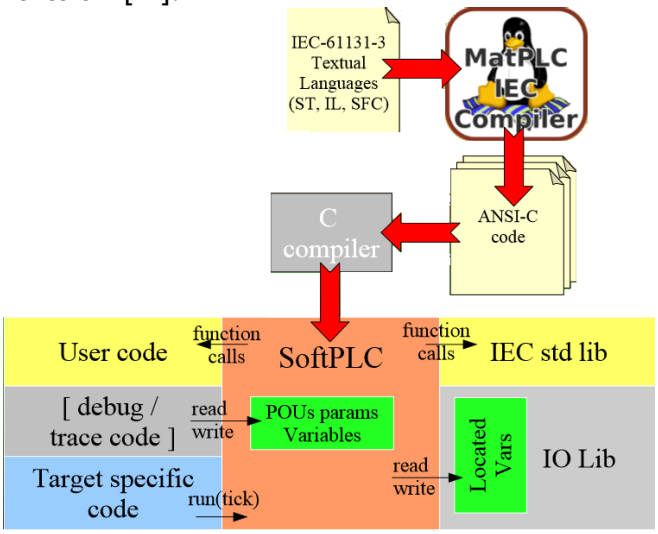

Figura 13 - Etapas gerais de compilação e organização do código [59]

4.1.1.4 Plugins

A única forma do *SoftPLC* comunicar com dispositivos externos associados ao projecto a controlar é através dos *plugins* que podem ser adicionados. Importa enquadrar o uso dos plugins com o disposto na norma IEC 61131-3. Está definida uma simbologia própria para a associação de variáveis a dispositivos físicos ou lógicos na estrutura de entrada, saída ou memória do PLC. Essa simbologia é composta pela concatenação de:

'%' + localização + tamanho + um ou mais inteiros separados por '.'

Os prefixos para a localização e para o tamanho podem ser consultados na Tabela 3.

Tabela 3 - Prefixos para a localização e tamanho das directly represented variables [5]

Tipo	Prefixo	Significado
Localização	I	Entrada
	Q	Saída
	M	Memória
Tamanho	X ou vazio	1 bit
	B	8 bits
	W	16 bits
	D	32 bits
	L	64 bits

Actualmente estão disponíveis dois plugins: *CanFestival* e *SVGUI*. O *CanFestival* permite a comunicação com dispositivos externos utilizando o protocolo de comunicação *CanOpen*. Já o *SVGUI* tem como objectivo fornecer ao utilizador uma interface HMI com o objectivo de supervisionar e controlar o estado de um determinado processo, tal como um SCADA.

Apesar de não ter sido lançado para o público em geral devido a não estar totalmente concluído, existe também o *plugin Modbus* que permite a comunicação com outros dispositivos através do protocolo *Modbus*. Neste momento encontra-se apenas implementado o funcionamento do *Beremiz* como *Master* e, como ainda não é possível utilizar o protocolo como *Slave*, torna-se desta forma impossível exportar variáveis através do mesmo.

Uma vez que o protocolo *Modbus* ainda não se encontra totalmente implementado, não é também possível utilizar todas as suas funções, estando apenas disponíveis as seguintes funções:

- 01 – Read Coils
- 02 – Read Input Discretes
- 03 – Read Holding Registers
- 04 – Read Input Registers
- 15 – Write Multiple Coils
- 16 – Write Multiple Registers

4.2 Ferramentas para desenvolvimento de interfaces SCADA

Numa fase mais prematura da dissertação, procedeu-se ao teste de diferentes ferramentas para o desenvolvimento de SCADA's com o objectivo de encontrar aquela que melhores características apresentava para a realização de um programa para a monitorização da LPF. Assim, exploraram-se três ferramentas: *PVBrowser*, *Eclipse SCADA* e *ScadaBR*.

4.2.1 PVBrowser

4.2.1.1 Visão Geral

O PVBrowser é um *SCADA application framework* baseado em *web browsers* que providencia ao utilizador um browser especializado para o cliente (Figura 15 - pvbrowser) e um IDE (pvdevelop) para a criação de servidores (pvserver) que implementam a visualização à qual a aplicação cliente acede. Funciona com o princípio cliente/servidor e está sob a licença LGPL. O seu princípio de funcionamento é visível na Figura 14.

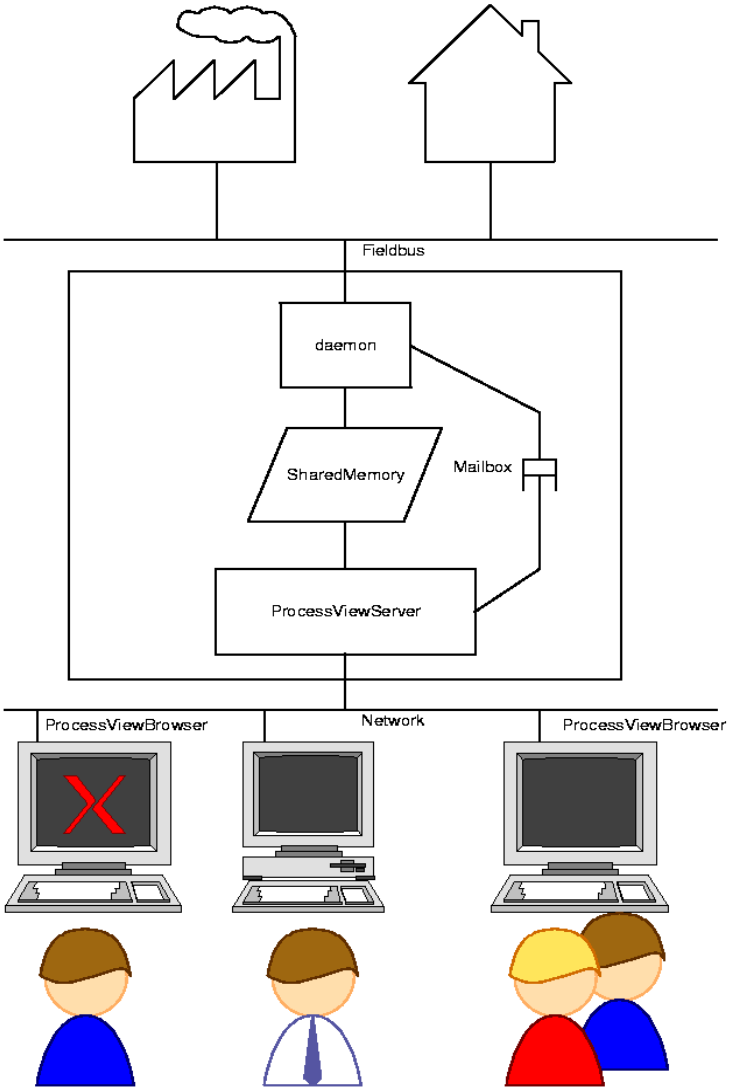

Figura 14 - Princípio de funcionamento do PVBrowser [43]

Este *software* tem a capacidade de fornecer programas para a aquisição de dados para uma grande variedade de protocolos de comunicação com a aplicação M2M. Os programas de aquisição de dados é feita na forma de *daemons* (programa de computador que roda de forma independente em *background*, ao invés de ser controlado directamente por um utilizador). O *daemon* lê ciclicamente as variáveis de um PLC ou de sistema *fieldbus* e armazena os seus

valores numa memória partilhada. Paralelamente, o *daemon* espera na sua *mailbox* por comandos de saída, que serão enviados para o PLC ou para o *fieldbus*.

A memória partilhada é memória RAM com diversos processos podem aceder ao mesmo tempo. A coordenação do acesso paralelo a esta memória feito por *mutex* (*mutual exclusion*), para evitar que dois processos tenham acesso ao mesmo tempo a um recurso partilhado.

A *mailbox* é um mecanismo de comunicação entre processos que estejam no mesmo computador. É organizada como *fifo* (first in first out). Vários processos podem escrever nela ao mesmo tempo mas esta pode ser lida apenas por um processo de cada vez.

O cliente *pvbrowser*, que estará a correr no computador do utilizador, pode ser comparado a um *web browser* mas, ao invés de fazer o *display* de páginas estáticas HTML, mostra *widgets,* que são objectos gráficos dinâmicos que mudam consoante o estado da variável. A aplicação cliente utiliza a biblioteca Qt4 que é um *framework* multiplataforma para desenvolvimento de interfaces gráficas em C++.

A aplicação servidor (*pvserver*) é feita pelo programador da visualização com a ajuda de um IDE que tem ambiente gráfico e ambiente para programar. O *pvserver* tem capacidades quase ilimitadas uma vez que em C/C++ é possível fazer as mais diversas coisas.

A comunicação entre o *pvbrowser* e o *pvserver*, que enviam comandos pela rede para serem interpretados do outro lado, é feita por TCP/IP. Os comandos consistem numa linha de texto e terminam com o símbolo de *newline*. Assim, o *pvserver* envia comandos ao *pvbrowser* que os irá transformar em chamadas aos métodos Qt. Ao mesmo tempo, o *pvbrowser* pode também enviar comandos quando determinado evento dispara.

O *layout* é todo feito graficamente através do IDE, tendo o utilizador que preencher as funções que lidam com os eventos (slot functions) no ambiente de programação.

O utilizador pode saltar facilmente de servidor para servidor bastando para isso aceder ao endereço correspondente, tal como é feito em *web browsers*. Caso haja alguma modificação na aplicação, ela apenas tem que ser feita no servidor, nunca tendo o utilizador que fazer nenhuma alteração no seu computador [43] [60].

4.2.1.2 Objectos gráficos e programação

No PVBrowser, todos os objectos gráficos fornecidos pela Qt podem ser utilizados. Para além destes objectos, existem ainda gráficos xy, maps de bits e mapas vectoriais. Também é possível a integração de programas externos de gráficos ou estatística ou de uma *webcam*, por exemplo.

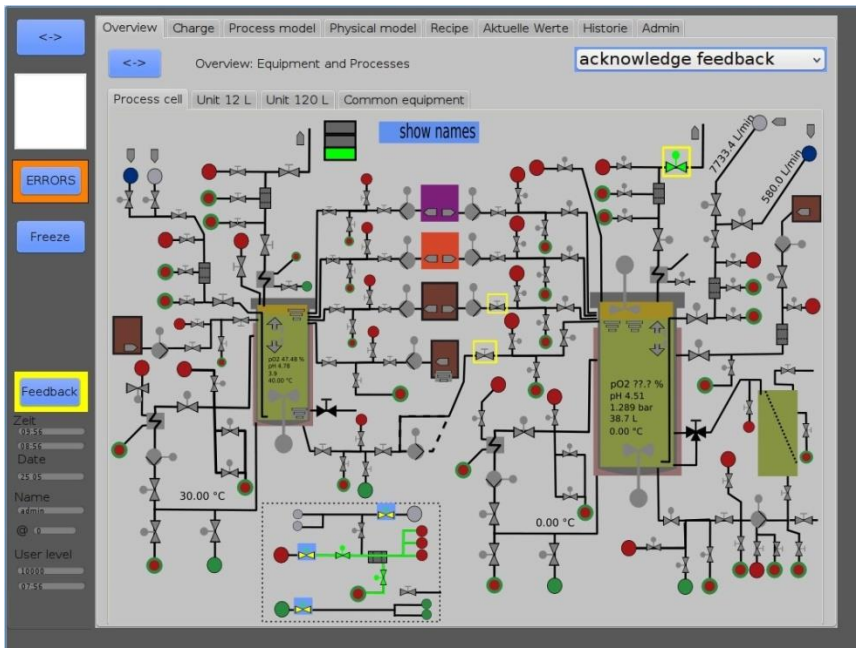

Figura 15 - Exemplo de um pvbrowser [61]

Caso o utilizador não queira fazer nada de transcendente, apenas terá que preencher as slot functions existentes, sendo ainda assim necessário o conhecimento da linguagem de programação C/C++.

Exemplos de slot functions preenchidas:

```
static int slotNullEvent(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  int i;
  const char *cptr;

  for(i=ID_MAIN_WIDGET+1; i<ID_END_OF_WIDGETS; i++)
  {
    cptr = toolTip[i];
    if(*cptr != '#' && *cptr > ' ') // if there is a variable name
    {
      pvPrintf(p,i,"%s=%s",cptr,opc.stringValue(cptr));
    }
  }
  pvPrintf(p,labelReadErrorCount,"readErrorCount=%d",opc.readErrorCount());
  pvPrintf(p,labelWriteErrorCount,"writeErrorCount=%d",opc.writeErrorCount());
  return 0;
}
```

```
static int slotButtonEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  int   ival;
  float fval;
  switch(id)
  {
    case buttonInt16:
      ival = opc.intValue("test/Int16");
      ival += 1;
      opc.writeIntValue("test/Int16",ival);
      break;
    case buttonInt32:
      ival = opc.intValue("test/Int32");
      ival += 1;
      opc.writeIntValue("test/Int32",ival);
      break;
    case buttonFloat:
      fval = opc.floatValue("test/float");
      fval += 1.0f;
      opc.writeFloatValue("test/float",fval);
      break;
    case buttonDouble:
      fval = opc.floatValue("test/double");
      fval += 1.0f;
      opc.writeFloatValue("test/double",fval);
      break;
    case buttonHello:
      opc.writeStringValue("test/string","hello");
      break;
    case buttonWorld:
      opc.writeStringValue("test/string","world");
      break;
    default:
      break;
  }
  return 0;
} [60]
```

4.2.1.3 Notas finais

É possível concluir que o PVBrowser é um *software* poderoso, com inúmeras potencialidades, sendo dos SCADA's *open source frameworks* mais utilizados no mundo. Apesar de todas as suas potencialidades, a utilização desta ferramenta está limitada a apenas alguns utilizadores uma vez que para a utilizar é necessário ter conhecimentos de programação em C/C++ ou em Lua já que, ao contrário de outros SCADA's, não é possível criar animações apenas através da interface gráfica. Apesar de ser possível adicionar elementos gráficos escolhendo-os de uma lista e definindo posteriormente as suas propriedades através do ambiente gráfico, para a elaboração de animações é necessário fazer funções numa destas duas linguagens de programação. Ou seja, apesar da sua capacidade de funcionar como um sistema SCADA, substituindo outras ferramentas cuja licença é comercial, o PVBrowser não *user friendly*.

Desta forma, como para a sua exploração eram necessários conhecimentos consistentes na programação C++ de maneira a implementar qualquer funcionalidade ao sistema, o que levaria a um tempo de aprendizagem, optou-se pela escolha de uma outra ferramenta já que durante o estado da arte se encontrararm diversos *softwares* diferentes com características interessantes que também havia interesse em explorar.

4.2.2 Eclipse SCADA

4.2.2.1 Visão Geral

O Eclipse SCADA é, como o nome diz, um sistema SCADA. Assim, o propósito deste sistema é o de colectar dados de vários sistemas, agregá-los, avaliar o seu estado e visualizar o mesmo. Para além destas funcionalidades, deve também providenciar ao utilizador uma maneira de controlar os sistemas conectados. Como é óbvio, é também importante armazenar dados de maneira a poderem ser avaliados mais tarde.

Este projecto surgiu quando o projecto *openSCADA*, criado em 2006, se moveu para a Eclipse. Apesar do *openSCADA* ter a licença LGPLv3, os seus contribuidores concordaram em alterar a licença para EPL (*Eclipse Public License* – é uma licença de software *open source* com protecção *copyleft* mais fraca que a licença GPL, sendo incompatível com a mesma; o utilizador de um programa sob a licença EPL pode utilizar, modificar, copiar e distribuir tanto o programa como versões modificadas [62]) [63]. O funcionamento do software *openSCADA*, está documentado no capítulo 3.3.

O Eclipse SCADA é independente da plataforma, ou seja, funciona tanto em *Windows*, como *Linux* ou *Mac OS X*. Conforme mostra a Figura 16, os componentes constituintes da sua arquitectura são os seguintes:

- DA (Data Access) – A aquisição deve poder ser feita em tempo quase real.

- AE (*Alarms & Events*) – Monitorização dos dados provenientes do sistema DA e armanezar os resultados da monitorização.
- HD (*Historical Data*) – Armazenar os valores dos dados fornecidos pelo sistema DA [64].
- CA (*Configuration*) – Criação de configurações
- VI (*GUI/HMI*) – Criação dos elementos da interface gráfica. É baseado em Draw2D.

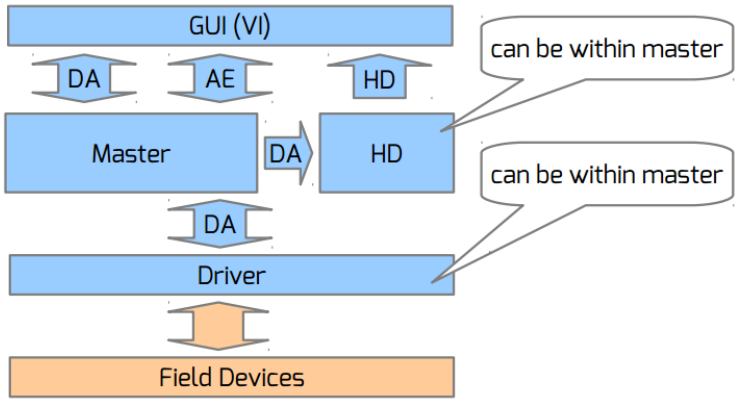

Figura 16 - Arquitectura do Eclipse SCADA

O Eclipse SCADA ainda se encontra em desenvolvimento, estando previsto o lançamento de uma versão mais completa (v0.2.0) no fim do presente ano, não sendo esta ainda a versão final do programa [65].

4.2.2.2 Notas finais

Inicialmente, este tinha sido o *software* escolhido para a realização do programa SCADA para a monitorização da LPF. Apesar de se ter chegado a instalar esta ferramenta, deparou-se com um problema na criação de um projecto, que é algo complexo uma vez que é necessário utilizar o IDE eclipse juntamente com o Eclipse SCADA. Esse problema impediu a continuação da exploração desta ferramenta pois nao se conseguiu fazer a ligação com o simulador. Depois de bastante tempo perdido, tal como o que aconteceu com o *software* PVBrowser, optou-se pela escolha de outra ferramenta. Isto sucedeu quando se verificou que cada problema que aparecesse com este *software* teria dificuldades acrescidas pois não se encontraria qualquer documentação que servisse de ajuda. Assim, depois de alguns dias perdidos na configuração da ligação *Modbus* com o simulador da LFP, optou-se por abandonar a exploração desta ferramenta.

4.2.3 ScadaBR

4.2.3.1 Visão Geral

O ScadaBR é uma aplicação multiplataforma sob a licença GNU/GPL baseada em *Java*, ou seja, computadores com o *Windows*, *Linux* e outros sistemas operativos instalados podem executar o *software* a partir de um servidor de aplicações (*Apache Tomcat*), incluído nas versões mais recentes. O projecto nasceu na MCA Sistemas em Florianópolis/SC, sendo activamente desenvolvido na Fundação CERTI por uma equipa de profissionais, em parceria com mais duas empresas da região, a Unis Sistemas e a Conetec [40]. Este software serve para automatizar processos de medição e automação, ou seja: através do ScadaBR o utilizador pode aceder e controlar dispositivos físicos como sensores, motores e outros tipos de máquinas. Entre outras funcionalidades, é possível guardar dados dos sensores continuamente numa base de dados, visualizar os históricos, e também receber alarmes, controlar o processo através de *scripts*, etc.

O programa necessita da instalação do *Java 6* (JDK 1.6) e do *Apache Tomcat 6*, o qual permite o acesso ao programa como se fosse uma página *Web*, através de um *browser*, acedendo ao endereço localhost:8080/ScadaBR (8080 é a porta pré-definida do *Apache Tomcat*, podendo ser alterada pelo utilizador) e utilizando *admin* como utilizador e como password. Devem também ser instaladas as bibliotecas RxTx para o Java.

Para executar o ScadaBR, este deve ser acedido a partir de um browser, preferencialmente o *Firefox* ou o *Chrome*. A interface principal do ScadaBR é de fácil utilização e oferece visualização das variáveis, gráficos, estatísticas, configuração dos protocolos, alarmes, construção de representações gráficas tipo HMI e uma série de opções de configuração.

Após configurar os protocolos de comunicação com os equipamentos e definir as variáveis (entradas e saídas, ou "tags") de uma aplicação automatizada, é possível fazer representações gráficas utilizando o próprio navegador. Também é possível criar aplicações personalizadas, em qualquer linguagem de programação moderna, a partir do código-fonte disponibilizado ou da sua API "web-services".

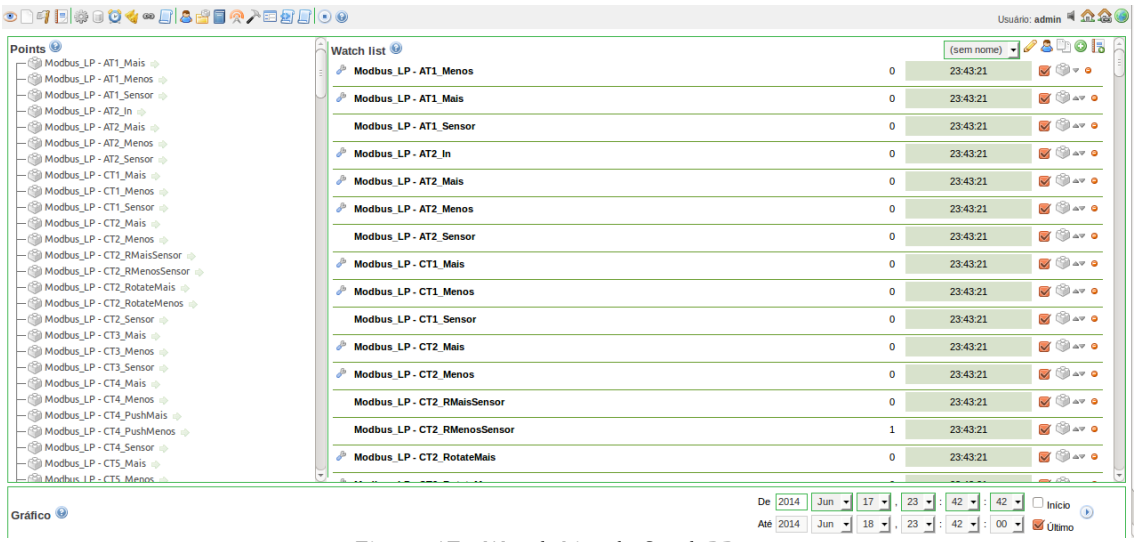

Figura 17 - Watch List do ScadaBR

Na parte superior da Figura 17 é possível vizualizar a barra de ferramentas do ScadaBR. Toda a navegação deste *software* é feito através desta barra, onde o utilizador pode aceder a todos os menus: *data sources*, representação gráfica, relatórios, opções do programa, importar/exportar projectos, etc.

Nos próximos subcapítulos é possível aceder a uma visão mais aprofundada sobre os aspectos mais importantes deste sistema SCADA.

4.2.3.2 Data Sources e Data Points

Os *data sources* são, como o nome indica, as fontes de dados recebidas pelo sistema. Ou seja, é através das *data sources* que se faz a configuração dos equipamentos físicos que serão controlados. Há diversos protocolos de comunicação suportados pelo ScadaBR, entre eles o estão: *Modbus*, HTTP, TCP/IP, SQL, OPC, entre outros. É ainda possível a adição de novos protocolos pelo utilizador.

Os *data points* são os pontos de medição, isto é, as variáveis. Armazenam todos os valores que são adquiridos pelo *data source*. Nas propriedades do *data point* (Figura 18) podem definir-se várias coisas, como o seu nome, o seu endereço, qual o seu tipo de dados, quanto tempo é que deve ser guardado no histórico ou qual o tipo de gráfico que deve exibir os valores medidos. O ScadaBR suporta quatro tipos de dados:

- Binários: 0 ou 1;
- Numéricos: valores de temperatura ou pressão, por exemplo;
- Alfanuméricos: sequência de caracteres;
- Multiestados: ligado/desligado/inactivo/activo, etc.

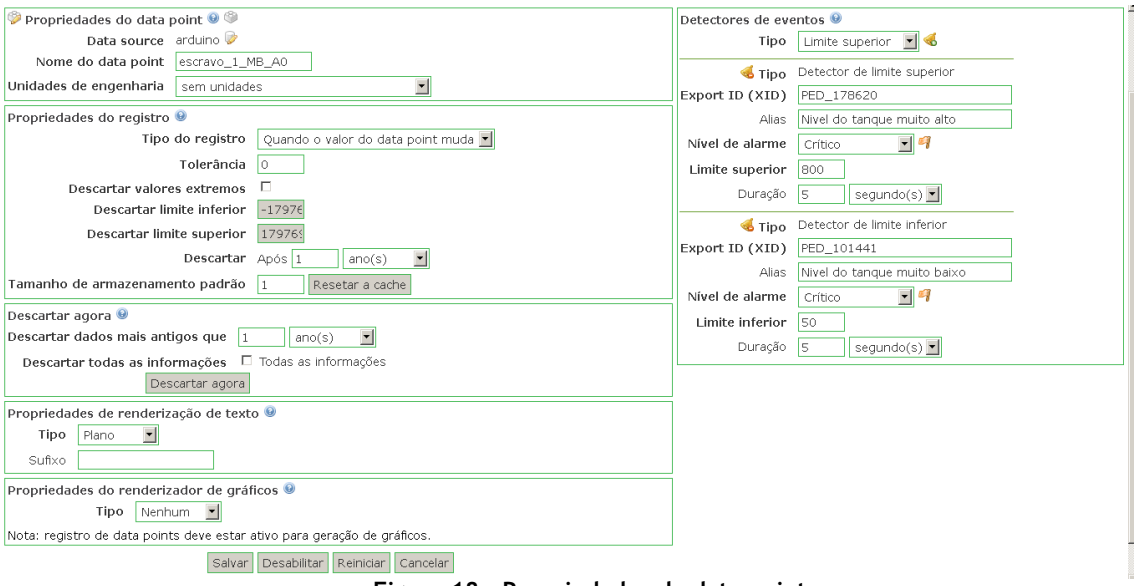

Figura 18 - Propriedades do data point

4.2.3.3 Watch List

Watch Lists (ver Figura 17) são listas dinâmicas que contêm os valores dos *data points* definidos pelo utilizador actualizados em tempo real. Pode ainda visualizar-se um gráfico de tendências instantâneas com todas as variáveis que o utilizador tiver adicionado à *watch list*.

4.2.3.4 Representações gráficas

As representações gráficas, também conhecidas por HMI, servem para o utilizador poder monitorizar os *data points* em tempo real. Para essa monitorização o utilizador pode adicionar, através da funcionalidade *drag and drop*, elementos gráficos variados como gráficos, ícones, botões e algumas imagens pré-definidas que mudarão de estado conforme o comportamento da variável. Os elementos gráficos devem ser colocados sobre uma imagem de fundo definida previamente pelo utilizador.

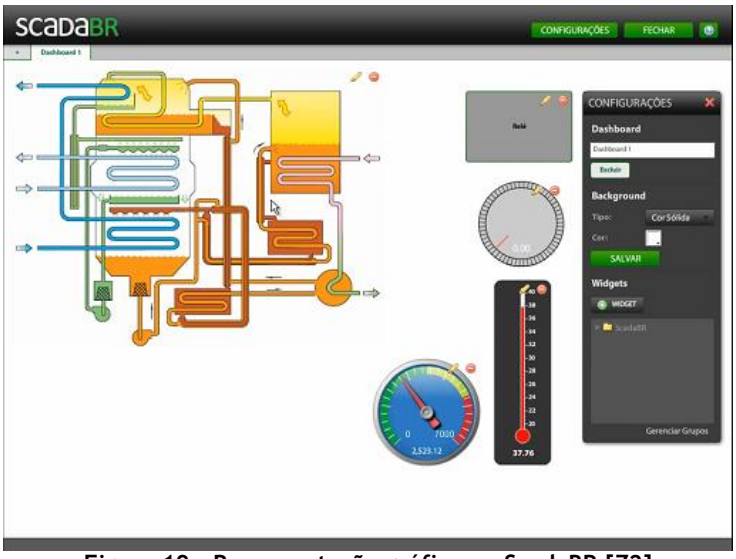

Figura 19 - Representação gráfica no ScadaBR [72]

4.2.3.5 Eventos e Alarmes

O ScadaBR aguenta cerca de 500 eventos por segundo [71], sendo que um evento é a ocorrência de uma determinada condição. No ScadaBR há eventos do sistema (inicialização do sistema, *login* de utilizadores, erros do *data source*, etc.) e eventos definidos pelo utilizador, como detectores de valor, eventos que ocorrem depois de um determinado tempo e eventos compostas por condições que envolvem mais do que variável, por exemplo. Essas condições podem criar alarmes, que normalmente exigem a visualização e reconhecimento por parte do utilizador Os alarmes têm um grau associado que vai desde o nível informação até ao nível *life safety*. Quando um alarme dispara, o utilizador ouve um alerta sonoro, caso assim esteja definido, e terá um aviso na parte superior do *screen* a piscar até que o alarme seja reconhecido .

A diferença entre eventos e alarmes é que os eventos não pedem nenhuma acção aos utilizadores nem tem um nível associado.

4.2.3.6 Relatórios

Os relatórios (exemplificado na Figura 20) servem para o utilizador ter um visão sobre o histórico de uma variável através de gráficos com o seu comportamento durante um determinado período, dados estatísticos e histórico de dados. Podem ser agendados para ser gerados automaticamente e podem também ser enviados por e-mail.

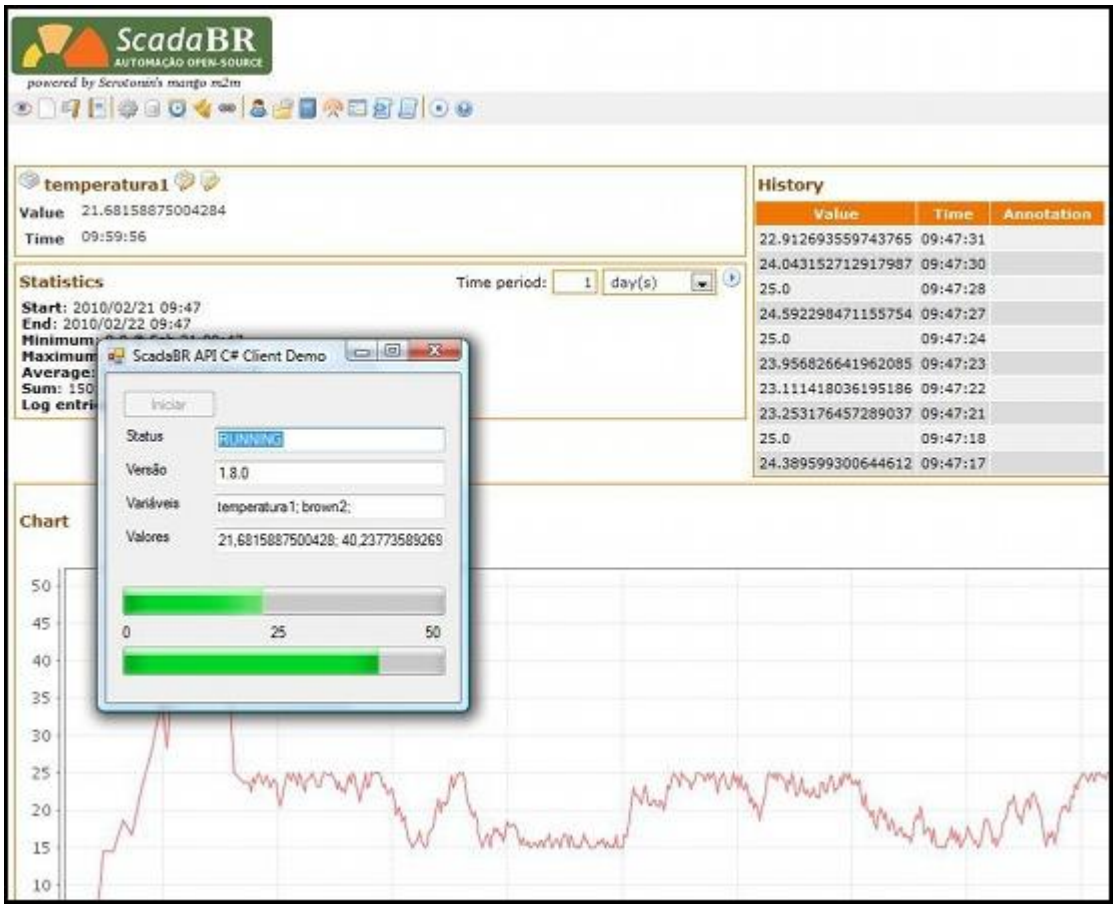

Figura 20 - Relatório ScadaBR [66]

4.2.3.7 Scripts

Os *scripts* servem para a criação de lógica como por exemplo: operações de aritmética, ciclos *while*, *if*, fazer um contador, etc. De maneira a utilizar os scripts, o utilizador deve criar um *meta data source* que terá o seu valor definido consoante o que estiver definido no script, que pode ser dependente do valor de alguns *data points*. A linguagem que se utiliza na escrita dos *scripts* é *javascript*.

4.2.3.8 API

A API - conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades por aplicações que não pretendem envolver-se em detalhes da implementação do software, mas apenas utilizar seus serviços [67] - permite que o utilizador adicione funcionalidades ao sistema.

A API do ScadaBR é um web-service que utiliza a tecnologia SOAP que é um protocolo multiplataforma para comunicação entre aplicações, cujo formato da mensagem é baseado em XML, que comunica via internet [68]. Isto permite estender o ScadaBR através da arquitetura "cliente-servidor", onde o ScadaBR age como um servidor, e um módulo externo (cujo desenvolvimento é personalizado) actua como cliente. O ScadaBR e a aplicação cliente podem ser executadas na mesma máquina, ou em computadores separados, desde que

conectados e acessíveis via IP. Para utilizar a API, deve-se utilizar a infra-estrutura de acesso a *Web-Services* da linguagem escolhida para desenvolvimento. Normalmente, isto resume-se a utilizar uma biblioteca de acesso ao SOAP, seguindo a sua documentação específica. Na maioria das linguagens, especialmente as que têm suporte mais automatizado ao SOAP (como Java e .NET), realizam-se apenas dois passos:

1. Criação das classes de API necessárias, a partir da importação do arquivo WSDL (web-services description language);
2. Acesso aos métodos da API.

A API SOAP do ScadaBR expõe as funcionalidades mais importantes do SCADA, que são a leitura e escrita de *tags*, a consulta ao histórico de dados e o acesso a alarmes e eventos, entre outos. A API pode ser estendida conforme as necessidades específicas do utilizador.

Entre os métodos atualmente disponíveis, os mais utilizados são:

- getStatus = retorna o estado de operação do ScadaBR
- browseTags = lista as tags disponíveis
- readData = retorna a leitura de uma ou mais variáveis (Tags)
- writeData = escreve valores em Tags
- writeStringData = escreve valores em Tags (para clientes sem compatibilidade com o tipo AnyData)
- getDataHistory = retorna o histórico de uma tag
- getActiveEvents = retorna os eventos ativos , como por exemplo os alarmes
- getEventsHistory= retorna o histórico de eventos, mesmo os inactivos
- ackEvents = reconhece um evento/alarme
- browseEventsDefinitions= lê os tipos de eventos definidos
- annotateEvent = adiciona uma mensagem de utilizador a um evento.

Alguns métodos permitem uma manipulação mais detalhada das configurações do ScadaBR, como:

- configureDataPoint
- removeDataPoint
- browseDataPoints
- removeDataSource
- configureDataSource
- browseDataSources[69]

Capítulo 5

Teste e Validação

5.1 Programa de controlo

5.1.1 Linha de Produção Flexível

Para o teste e validação da ferramenta de controlo escolhida (*Beremiz*) para o projecto da dissertação, fez-se um programa capaz de controlar o simulador da Linha de Produção Flexível (LPF) existente no Departamente de Engenharia Electrotécnica da FEUP. A LPF e o simulador podem ser visualizados nas figuras 21 e 22, respectivamente.

A LPF foi criada e desenvolvida pela marca STAUDINGER, tendo sido projetada e desenhada de acordo com os requisitos exigidos pela FEUP [70]. Encontra-se dividida em 4 módulos:

- Armazém
- Zona de montagem
- Zonas de maquinação
- Zonas de carga e escarga

Neste projecto, decidiu-se não ser necessária a utilização dos *pushers*. Decidiu-se também que os motores das máquinas e do robot não seriam utilizados uma vez que isso não traria nenhuma vantagem para a validação da ferramenta pois utilizando os motores de todos os tapetes lineares e rotativos, bem como os seus sensores, ficaria patente a capacidade ou não do *Beremiz* controlar este dispositivo. A disposição dos tapetes no simulador é visível na Figura 23.

Assim, definiu-se que a LPF se iria comportar da seguinte maneira: a introdução de peças deve ser feita no tapete AT1 (mesmo sendo possível colocar peças em qualquer tapete) e o objectivo é que essa mesma peça vá parar ao tapete AT2. Por definição, a peça fará o caminho mais curto possível. Caso os tapetes do caminho mais curto estejam ocupados, ao invés de esperar, a peça deve ser capaz de efectuar outro caminho. A LPF deve ter a capacidade de funcionar com tantas peças quantas as que o utilizador quiser, sendo possível ao utilizador introduzir uma peça no tapete AT1 mal este se encontre sem nenhuma peça.

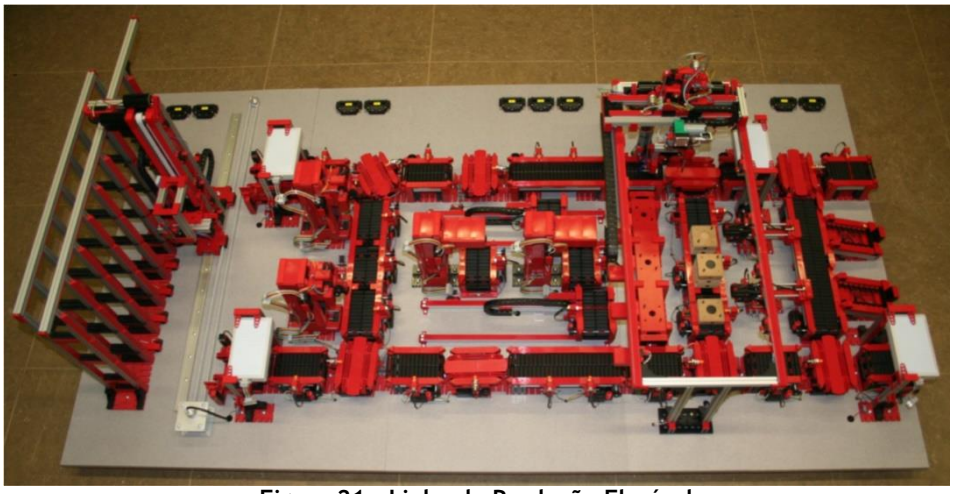

Figura 21 - Linha de Produção Flexível

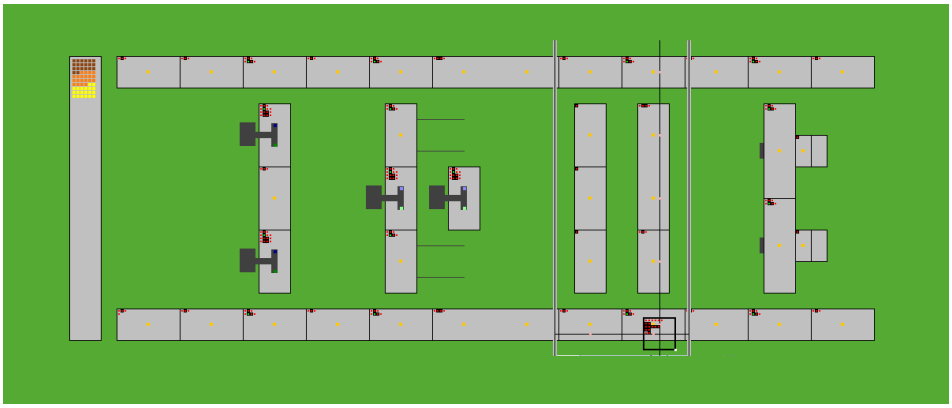

Figura 22 - Simulador da Linha de Produção Flexível

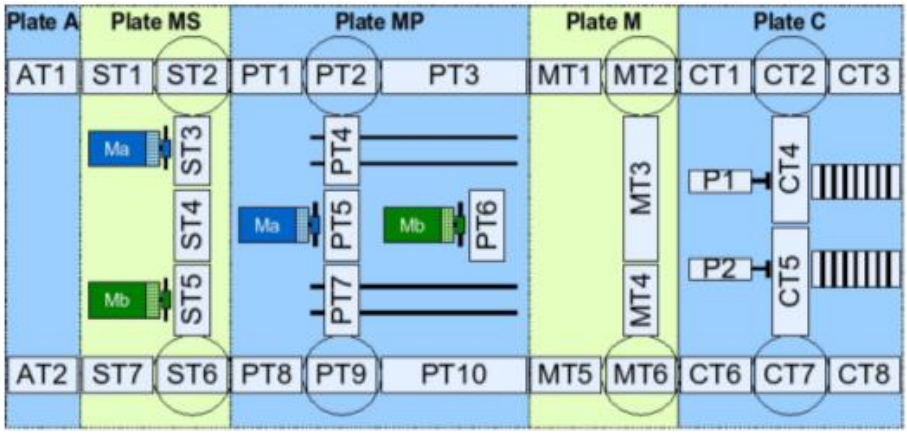

Figura 23 - Disposição dos tapetes no simulador

5.1.2 Comunicação

Para o programa de controlo se conectar à LPF foi necessário estabelecer uma ligação através de um protocolo de comunicação. O *Modbus TCP* foi o protocolo escolhido, uma vez que é o único protocolo sem fios suportado pelo *Beremiz* (apenas suporta os protocolos *Modbus* e *CanOpen*), sendo também suportado pelo simulador da LPF. Para além disso, o

protocolo *Modbus* é um protocolo de comunicação *open source*, adaptando-se perfeitamente ao pretendido nesta dissertação.

Assim, o primeiro passo para a elaboração do programa foi a adição de um *Modbus Master* ao projecto criado no *Beremiz*. Definiu-se o IP como 127.0.0.1 (*localhost*), uma vez que o simulador corre no mesmo computador que estará a fazer o trabalho do PLC, correndo a aplicação criada no *Beremiz*. A porta definida foi a 5502 (definida tanto no Beremiz como nas opções do simulador).

Uma vez adicionado o *Modbus Master*, foi necessário adicionar dois *requests*: um capaz de ler os sensores (*inputs*) e outro capaz de escrever (editar o seu valor) nos motores(*outputs*).

Não foi possível adicionar um *Modbus Slave* uma vez que esta funcionalidade ainda não está implementada. Esta facto faz com que seja impossível exportar variáveis do *Beremiz*, não sendo assim possível mostrar o contador das peças que chegaram ao armazém na aplicação SCADA realizada para complementar esta aplicação de controlo.

Concluída a adição do protocolo *Modbus*, passou-se à declaração das variáveis.

5.1.3 Variáveis

Para fazer o programa, é necessário primeiro declarar as variáveis que serão utilizadas no mesmo. Decidiu-se que seria mais vantajosa a declaração de todas as variáveis como variáveis globais, utilizando-as como variáveis externas nos POU's definidos. Desta forma, não é necessária a introdução da mesma variável e consequente definição do seu endereço cada vez que seja necessário utilizá-la num novo POU.

Assim, recorrendo ao ficheiro io.xls do simulador, referente a todos os *inputs* e *ouputs* existentes no mesmo (e, consequentemente, na LPF) declararam-se todas as variáveis a serem utilizadas neste projecto bem como o seu endereço (para ser possível a comunicação *Modbus* entre o *Beremiz* e o simulador, permitindo a leitura e escrita dos valores das variáveis).

5.1.4 POU

Criadas as variáveis, passou-se à criação dos POU's cujo objectivo era controlar a LPF. Uma vez que existem apenas quatro tipos de tapete (tapete linear, tapete rotativo com duas entradas e uma saída, tapete rotativo com uma saída e duas entradas e tapete de descarga no armazém) decidiu-se pela criação de quatro *function blocks*, onde cada um destes FB's define o comportamento que cada tipo de tapete deve ter. O objectivo da criação destes quatro FB's é o de criar um POU com a linguagem FBD onde serão adicionados tantas instâncias desses FB's quantos os tapetes existentes na LPF. Assim, nesse POU, os FB's devem estar organizados com a mesma disposição dos tapetes da linha, sendo posteriormente ligados entre si para que a LPF funcione da forma pretendida, que está resumidamente explicada no secção 5.1.1. A elaboração de um programa com estas características ajuda o utilizador pois

torna-se fácil perceber o funcionamento do programa, facilitando o *debug* do mesmo. Para além disso, ao serem definidos apenas 4 FB's e instanciando-se estes mesmos FB's tantas vezes quantas as necessárias, o utilizador torna o seu programa consistente pois todos os tapetes do mesmo tipo irão funcionar exactamente da mesma forma. Também a correcção de erros se torna muito menos trabalhosa pois, para os corrigir, é apenas necessário corrigir um FB, ficando automaticamente todas as instâncias desse FB corrigidas, poupando tempo e trabalho ao utilizador, para além de diminuir a possibilidade da ocorrência de erros do mesmo.

As duas imagens seguintes são iguais, diferindo apenas no seu tamanho. Na Figura 24 é possível verificar que a disposição dos *function blocks* está feita de forma semelhante à disposição dos tapetes na LPF. Já na Figura 25, pode-se verificar como se interligam os *function blocks* entre si e como se ligam as variáveis a estes.

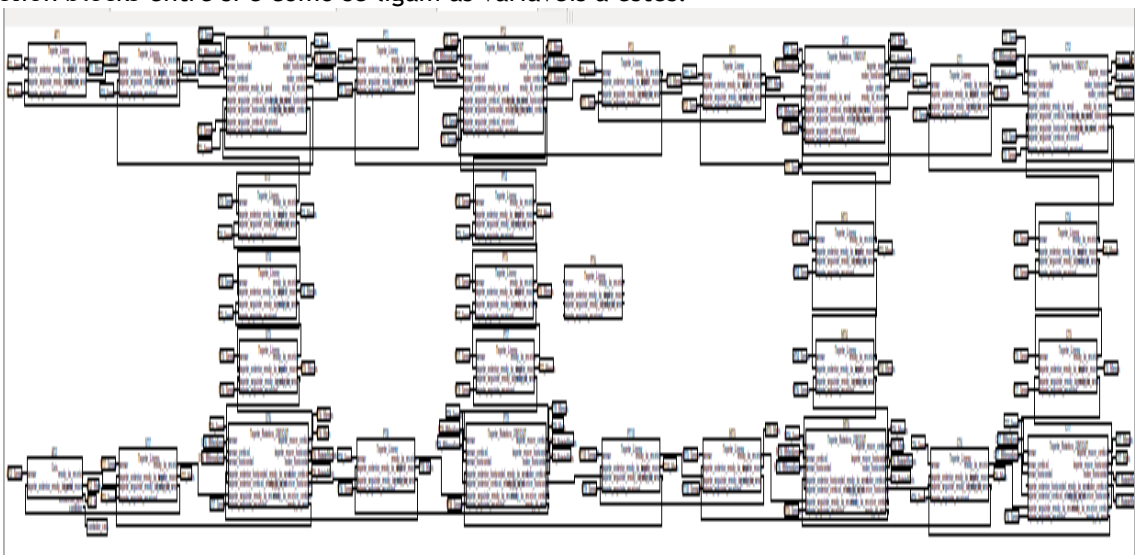

Figura 24 - FBD Program

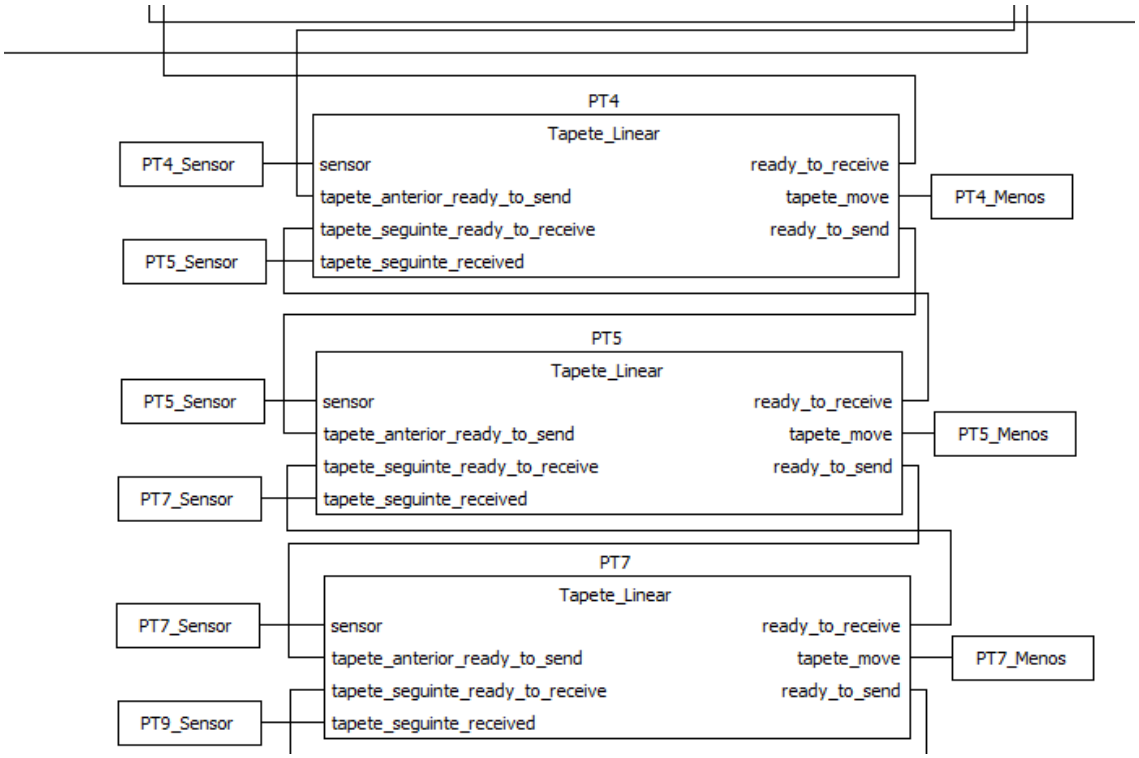

Figura 25 - Ligação entre variáveis e FB's

5.1.5 Function Blocks

Tal como explicado na secção anterior, elaboraram-se quatro *function blocks*, um para cada tipo de tapete existente na LPF.

O FB "Tapete Linear" (Figura 26) tem o sensor, tapete_anterior_ready_to_send, tapete_seguinte_ready_to_receive e tapete_seguinte_received como entradas e ready_to_receive, tapete_move e ready_to_send como saídas.

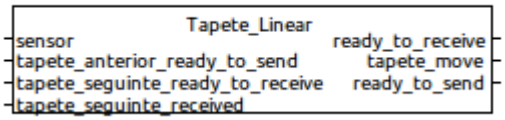

Figura 26 - Function Block Tapete Linear

A linguagem deste Function Block é SFC e o seu funcionamento está demonstrado na Figura 27.

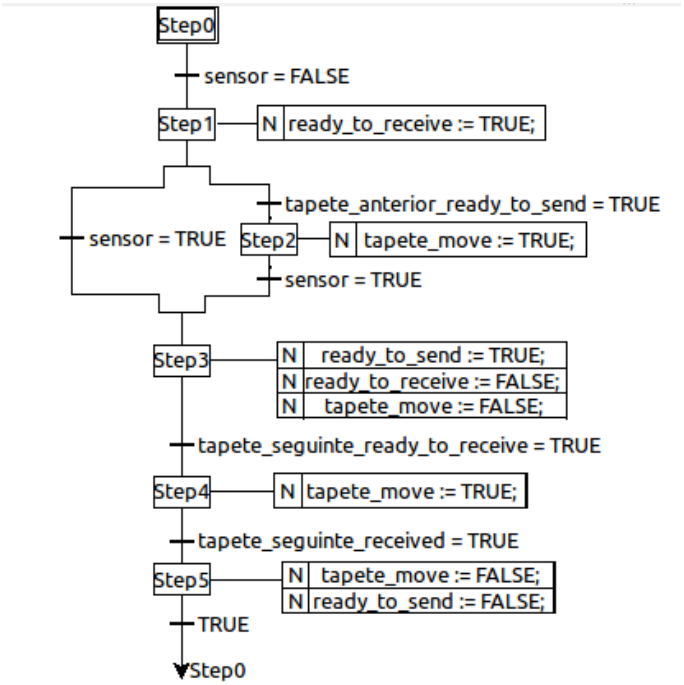

Figura 27 - Tapete Linear

As entradas do FB "Tapete Rotativo com 1 entrada e 2 saídas" (Figura 28) são o sensor, sensor_horizontal, sensor_vertical, tapete_anterior_ready_to_send, tapete_seguinte_vertical_ready_to_receive, tapete_segeuinte_horizontal_ready_to_receive , tapete_seguinte_vertical_received e tapete_seguinte_horizonal_received. Já as saídas são tapete_move, rodar_horizontal, rodar_vertical, ready_to_receive, ready_to_send_horizontal e ready_to send_vertical.

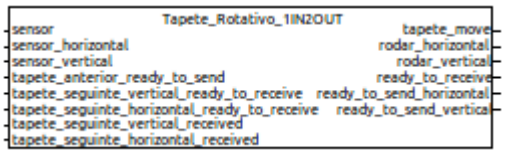

Figura 28 - Function Block Tapete Rotativo 1 IN 2 OUT

A linguagem deste FB é também SFC e na Figura 29 é possível ver qual o seu comportamento.

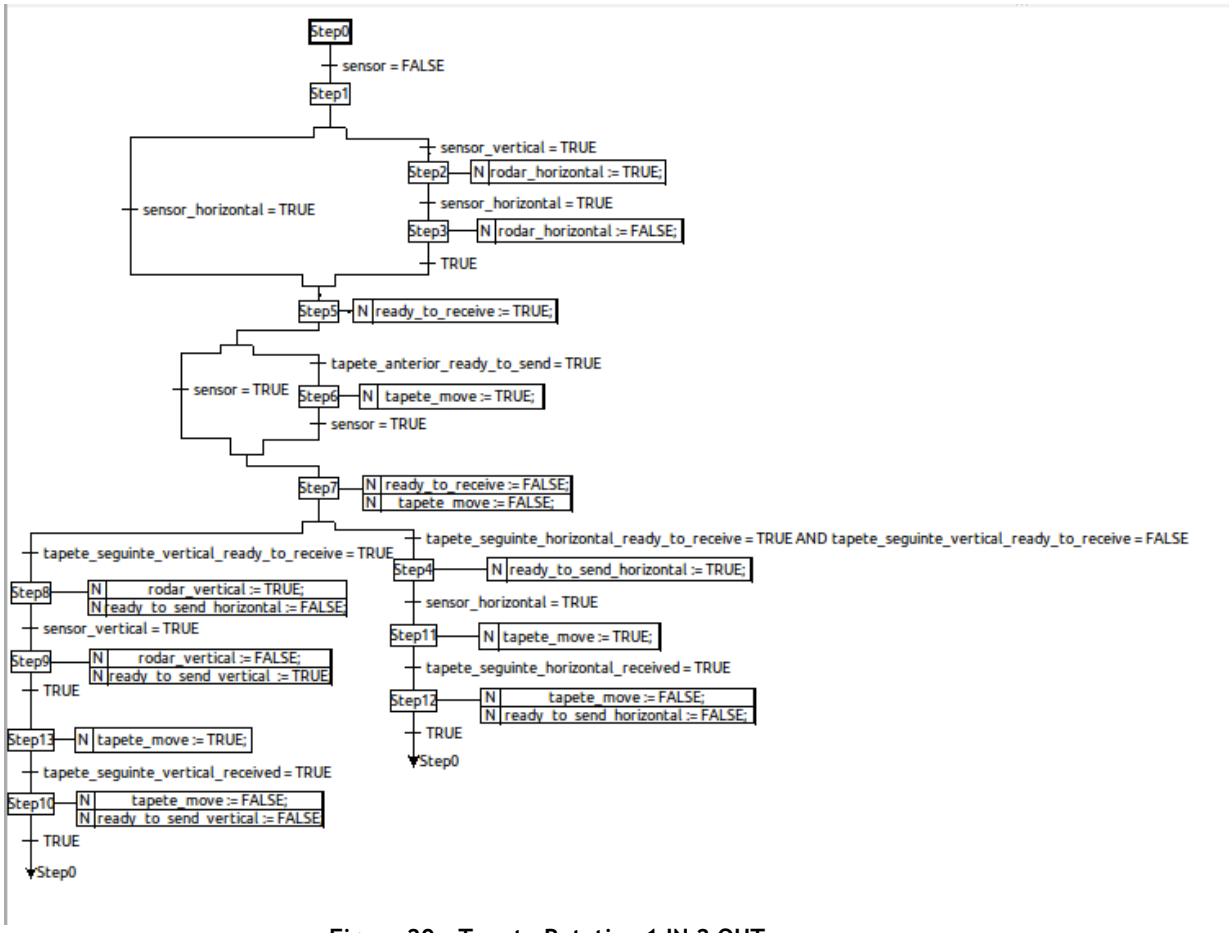

Figura 29 - Tapete Rotativo 1 IN 2 OUT

Já no FB "Tapete Rotativo com 2 entradas e 1 saída" (Figura 30), as entradas que o constituem são o sensor, sensor_vertical, sensor_horizontal, tapete_anterior_horizontal_ready_to_send, tapete_anterior_vertical_ready_to_send, tapete_seguinte_ready_to_receive e tapete_seguinte_received. As suas saídas são tapete_move_vertical, tapete_move_horizontal, rodar_horizontal, rodar_vertical, ready_to_receive_horizontal, ready_to_receive_vertical e ready_to_send.

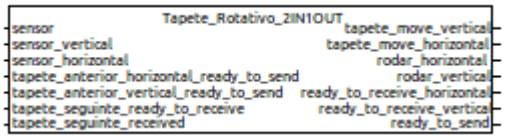

Figura 30 - Function Block Tapete Rotativo 2 IN 1 OUT

Tal como os FB's expostos anteriormente, também este tem a linguagem SFC. É possível ver como se comporta este *function block* na Figura 31.

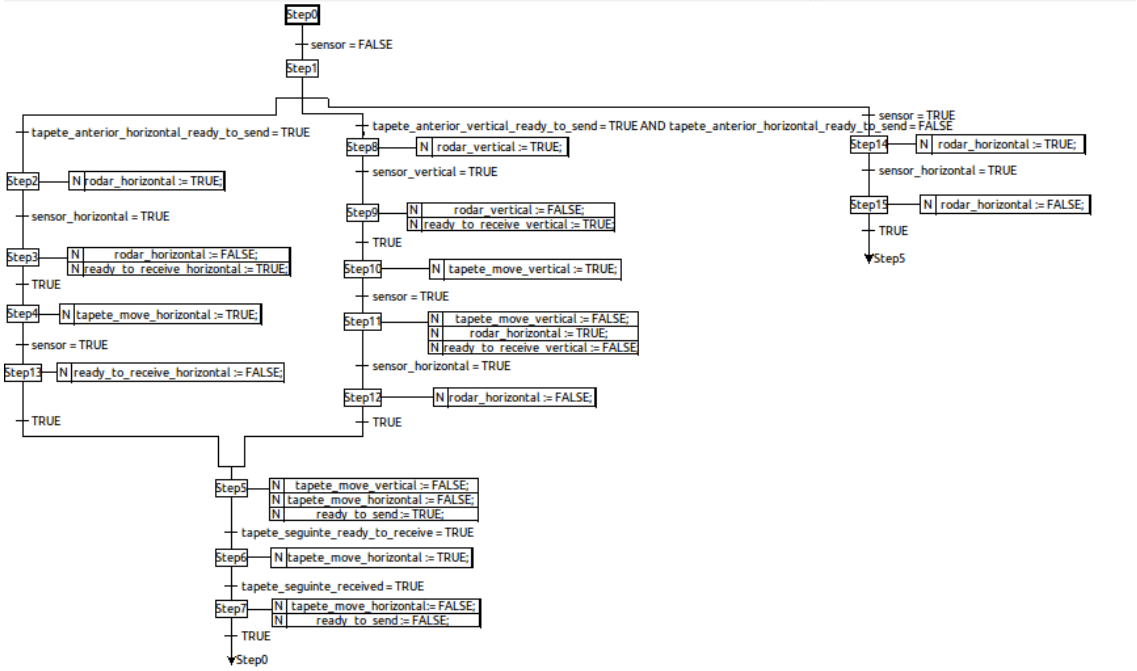

Figura 31 - Tapete Rotativo 1 IN 2 OUT

Por fim, o FB "Cais" (Figura 32) tem apenas 2 entradas, sendo elas o sensor e o tapete_anterior_ready_to_send. Como saídas este FB tem o contador, ready_to_receive, tapete_move e armazenar_peca.

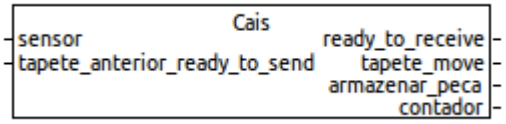

Figura 32 - Function Block Cais

Para compreender o funcionamento deste Function Block basta atentar na Figura 33.

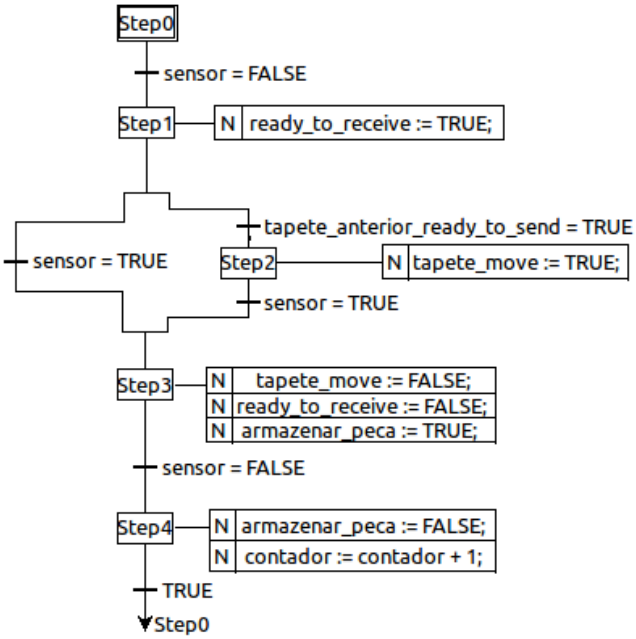

Figura 33 - Cais

5.1.6 Configurações

Foi necessária a criação de uma tarefa, definindo-se esta como cíclica e sendo o seu período de 10 milisegundos.

Por fim, de maneira se a poder transferir o programa para o PLC e colocá-lo no modo *run* criou-se uma nova instância associando o programa criado anteriomente a esta tarefa.

5.2 Testes Realizados ao software Beremiz

Para poder validar o *software Beremiz*, decidiu-se realizar testes a todos os recursos que este *software* possui. Assim, foi possível verificar a existência de *bugs* cujo relatório foi feito no subcapítulo 5.3 e tem o objectivo de obter uma correcção para os mesmos no futuro, tornando o *software* mais robusto. Apenas se realizaram testes para as linguagens ST (*Structered Text*), SFC (*Sequence Flow Chart*) e FBD (*Function Block Diagram*). As linguagens LD (*Ladder Diagram*) e IL (*Instruction List*) não foram testadas.

Todos os testes realizados estão enumerados no presente subcapítulo.

Variáveis

1. Criação de variáveis do tipo *elementary data types*:
    a. BOOL
    b. SINT
    c. INT
    d. DINT
    e. LINT
    f. USINT
    g. UINT
    h. UDINT
    i. ULINT
    j. REAL
    k. LREAL
    l. TIME
    m. DATE
    n. TIME_OF_DAY
    o. DATE_AND_TIME
    p. STRING
    q. BYTE
    r. WORD
    s. DWORD
    t. LWORD

u. WSTRING

Descrição detalhada dos testes Variáveis.1 – Adicionaram-se variáveis a todos os tipos de POU existentes, às quais se deu um valor. De seguida, compilou-se o POU e verificou-se o valor da variável no *debugger* do programa *Beremiz*. (Ex: No teste Variables.1.a adicionou-se uma variável do tipo BOOL, à qual se deu o valor de TRUE, a todos os tipos de POU: *function*, *function block* e *program*. Concluído este passo, compilou-se cada um dos POU.)

2. Utilização de *Numeric literals*:

a. Integer literals (ex: 5, -5, +5)
b. Real literals (ex: -5.0 5.052)
c. Real literals with exponents (ex: 5.0E+2, 5.2e-5)
d. Base 2 literals (ex: 2#1010_1010)
e. Base 8 literals (ex: 8#377)
f. Base 16 literals (ex: 16#FF)
g. Boolean zero and one
h. Booleans FALSE and TRUE
i. Typed literals (ex: BOOL#1, DINT#5)

Descrição detalhada dos testes Variáveis.2 – Nestes testes, relacionados com as variáveis do tipo numérico (INT e os seus derivados), atribuíram-se valores com as diferentes nomenclaturas que a norma IEC61131-3 prevê. Para cada teste especificado pelas letras a,b,c,d,e,f,g,h e i, atribuiram-se os valores referidos nos exemplos dos mesmos. Estas variáveis foram adicionadas a diferentes POU's que, depois de terem as variáveis declaradas e com um valor atribuído, foram compilados. Os seus valores foram verificados no *debugger* do programa *Beremiz*.

3. Utilização de *Character string literals*:

a. Single-byte character strings (ex: '', 'a', '$R')
b. Double-byte character strings (ex: "", "a", "$R")

Descrição detalhada dos testes Variáveis.3 – Estes testes são referentes às variáveis do tipo STRING e suas derivadas. Para estes testes, adicionaram-se variáveis do tipo *single byte character string* e *double byte character string*. Não se limitou a utilizar uma simples letra para estes tipos de variáveis, testando-se também o código referente a outros caracteres como *Line Feed* ($L) e *Carriage Return* ($R). O POU foi compilado e o valor das variáveis foi verificado no *debugger* do programa *Beremiz*.

4. Utilização de *Time literals*:

a. Time literal features (ex: T#10ms, t#-10.5s, TIME$10.5s, time#10h_10m_10.5s)

Descrição detalhada dos testes Variáveis.4 – Para o teste Variables.4, adicionou-se uma variável do tipo TIME a um POU à qual se deu valores utilizando diferentes nomenclaturas, tal como explicado no exemplo. Para cada nomenclatura do exemplo compilou-se o POU e verificou-se o valor da variável no *debugger* do programa *Beremiz*.

5. Utilização de *Date and time of day literals*:
   a. Date literals (ex: d#1990-05-25, DATE#1990-05-25)
   b. Time of day literals (ex: tod#10:00:00.50, TIME_OF_DAY#10:00:00.50)
   c. Date and time literals (ex: dt#1990-05-25-10:00:00.50, DATE_AND_TIME#1990-05-25-10:00:00.50)

Descrição detalhada dos testes Variáveis.5 – Adicionaram-se três variáveis, uma de cada tipo especificado neste conjunto de testes a um POU. Para cada uma destas variáveis atribuiram-se valores com diferentes nomenclaturas, conforme indicam os exemplos nas mesmas alíneas. De seguida, compilou-se o POU e verificou-se o valor das variáveis no debugger do programa Beremiz.

6. Criação de variáveis de diferente class:
   a. External
   b. Input
   c. Output
   d. Local
   e. Array

Descrição detalhada dos testes Variáveis.6 – Nestes testes adicionaram-se variáveis do tipo *Input* e *Output* a um FB e a uma função. Fez-se uma função simples para cada um destes POU's e, através da criação de um POU do tipo *program* ao qual se adicionaram variáveis locais, chamou-se a função e o FB criados utilizando as variáveis locais como argumentos. Após a compilação, verificou-se se o comportamento esperado se verificava. Uma vez testados as classes *Input*, *Output* e *Local*, procedeu-se à criação de variáveis externas do tipo BOOL em POU's do tipo *program* e FB. A variável local correspondente a essa variável externa foi declarada nas configurações do projecto à qual se atribuiu um endereço referente à comunicação através do protocolo *Modbus* com o simulador da LPF. Nos POU's criados para o teste das variáveis externas, fez-se um programa simples onde o valor da variável externa era alterado de TRUE para FALSE e de FALSE para TRUE de 5 em 5 segundos. Verificou-se o funcionamento através da ligação ao simulador no qual se constatou se a variável utilizada estava a alterar o seu valor conforme o definido nos POU's.

Funções

1. (ST) Criação de funções criadas pelo utilizador:
    a. Criação de uma função simples em ST que é chamada num programa ST.
    b. Criação de duas funções simples em ST em que um delas é chamada no programa ST. A funçao que é chamada chama a outra função criada previamente.

Descrição detalhada dos testes Funções.1 – Criou-se um programa A e duas funções, B (em que o *output* era a soma de 2 variáveis *input*) e C (em que o *output* era a multiplicação de 2 variáveis *input*). No programa declararam-se 2 variáveis e neste chamou-se a função B passando-lhe como parâmetros as 2 variáveis criadas. Pôde constatar-se que a função B realizou a soma devidamente. De seguida, no programa A chamou-se a função B que por sua vez chamou a função C. Foi possível ver que as variáveis foram sempre passadas correctamente e não houve qualquer problema em chamar uma função dentro de outra função. Para concluir, realizou-se um processo semelhante mas com a ordem trocada. Ou seja, no programa A chamou-se a função C que por sua vez chamou a função B. Mais uma vez foi possível verificar que o comportamento do *Beremiz* foi o esperado.

2. (FBD) Teste dos *Standard Function Blocks*:
    a. SR (SR bistable):
    b. RS (RS bistable)
    c. R_TRIG (Rising edge detector)
    d. F_TRIG (Falling edge detector)
    e. CTU, CTU_DINT, CTU_LINT, CTU_UDINT, CTU_ULINT (Up-Counter)
    f. CTD, CTD_DINT, CTD_LINT, CTD_UDINT, CTD_ULINT (Down-Counter)
    g. CTUD, CTUD_DINT, CTUD_LINT, CTUD_UDINT, CTUD_ULINT (UpDown-Counter)
    h. TON (On-delay timing)
    i. TOF (Off-delay timing)
    j. TP (Pulse timer)

3. (ST E FBD) Teste das *Numeric Functions*:
    a. ABS (absolute number)
    b. SQRT (square root)
    c. LN (natural logarithm)
    d. LOG (logarithm to base 10)
    e. EXP (exponentiation)
    f. SIN (sine)
    g. COS (cosine)

   h. TAN (tangent)
   i. ASIN (arc sine)
   j. ACOS (arc cosine)
   k. ATAN (arc tangent)

4. (ST E FBD) Teste das *Arithmetic Functions*:
   a. ADD (addition)
   b. MUL (multiplication)
   c. SUB (subtraction)
   d. DIV (division)
   e. EXP (exponent)
   f. MOD (modulo)
   g. MOV (move)

5. (ST E FBD) Teste das *Time Functions*:
   a. ADD_TIME
   b. ADD_TOD_TIME
   c. ADD_DT_TIME
   d. MULTIME
   e. SUB_TIME
   f. SUB_DATE_DATE
   g. SUB_TOD_TIME
   h. SUB_TOD_TOD
   i. SUB_DT_TIME
   j. DIVTIME

6. (ST E FBD) Teste das *Bitshift Functions*:
   a. SHL (Shift Left)
   b. SHR (Shift right)
   c. ROR (Rotate right)
   d. ROL (Rotate left)

7. (ST E FBD) Teste das *Bitwise Functions*:
   a. OR
   b. AND
   c. XOR
   d. NOT

8. (ST E FBD) Teste das *Selection Functions*:
    a. SEL
    b. MAX
    c. MIN
    d. LIMIT
    e. MUX

9. (ST E FBD) Teste das *Comparison Functions*:
    a. GT (greater than)
    b. GE (greater than or equal to)
    c. EQ (equality)
    d. LE (lower than or equal to)
    e. LT (lower than)
    f. NE (inequality)

10. (ST E FBD) Teste das *String Character Functions*:
    a. LEN (length of string)
    b. LEFT (string left to)
    c. RIGHT (string right to)
    d. MID (string in the middle of)
    e. CONCAT (concatenation)
    f. CONCAT_DAT_TOD (time concatenation)
    g. INSERT
    h. DELETE
    i. REPLACE
    j. FIND

Descrição detalhada dos testes Funções.2, Funções.3, Funções.4, Funções.5, Funções.6, Funções.7, Funções.8, Funções.9 e Funções.10 – Cada FB tem um tipo de dados definido para as suas entradas e para as suas saídas. Por exemplo o FB CTU tem 2 BOOL's (CU e R) e 1 INT (PV) como entradas e 1 BOOL (Q) e 1 INT (CV) como saídas. Para ver qual o funcionamento que cada FB teria que ter, consultou-se o livro [4] onde se pode ver quais as entradas e saídas de cada FB, bem como o que elas representam. Para além disso, neste livro é também visível uma representação do FB (a qual se comparou com a representação do FB no *Beremiz*) bem como a sua função escrita em ST. Desta forma, foi possível comprovar se os FB's e as suas variáveis no *Beremiz* correspondiam ao disposto na norma IEC 61131-3, sendo também possível verificar se o FB se estava a comportar conforme o que seria suposto, ou seja, se a saída do FB estava em conformidade com os valores das variáveis definidas como entrada. Realizou-se um teste para cada FB, criando um POU com a linguagem FBD onde se

adicionou um FB de cada vez bem como as variáveis necessárias ao seu funcionamento. Testou-se ainda a utilização de mais do que um FB de tipo diferente no mesmo POU. Todas estas funções, para além dos *Standard Function Blocks* (testados em Functions.2), foram também testados num POU com a linguagem ST. Todos os programas foram compilados e o valor das variáveis foi verificado no *debugger* do Beremiz e no simulador.

11. (FBD) Criação de novos *function blocks*, definidos pelo utilizador, e interligação dos mesmos.

Descrição detalhada dos testes Funções.11 – Para a criação do programa de automação referido no capítulo 5.1, foi necessário criar novos FB's que foram posteriormente instanciados tantas vezes quantas as necessárias e ligados entre si. Concluído este passo, compilou-se o POU do tipo FBD criado e verificou-se o funcionamento do mesmo através da visualização dos valores das variáveis no *debugger* do Beremiz.

12. (FBD) Utilização do EN (enable) e do ENO (enable output) em function blocks

Descrição detalhada dos testes Funções.12 – Adicinou-se um FB a um programa com a linguagem FBD. Nas opções desse FB colocou-se um visto na opção *execution control*. Se o valor da variável EN for FALSE ou 0, as funções do FB não devem ser executadas e o valor de ENO deve ser 0. De outra forma, o valor de ENO deve ser TRUE.

13. Teste de ST language statements:

a. ASSIGNMENT (:=)
b. RETURN
c. IF
d. CASE
e. FOR
f. WHILE
g. REAPEAT
h. EXIT

Descrição detalhada dos testes Funções.13 – Criou-se um POU com a linguagem ST e, depois de verificar no livro [4] qual a semântica associada a cada *statement*, criaram-se todos os diferentes *statements* especificados nas alíneas referentes a este teste. De seguida, compilou-se e verificou-se no *debugger* do *Beremiz* se o seu funcionamento era o esperado.

Sequence Flow Chart

1. Utilização de *Step Features*:

a. ***.X (step flag)
b. ***.T (step elapsed time)

Descrição detalhada dos testes Sequence Flow Chart.1 – Criou-se um POU do tipo programa onde a linguagem foi definida como SFC. Nesta criaram-se 2 transições diferentes que dependiam da etapa anterior, em que cada uma destas transições correspondia a um *step feature* diferente, representados nas alíneas a e b. Assim, criou-se um step inicial com o nome *step0*. A transição seguinte a este estado foi *step0*.X que estava ligada a uma nova etapa, o *step1,* e verificou-se que a transição se verificava sempre, tal como pretendido. De seguida, para verificar se o seu funcionamento estava mesmo correcto, negou-se esta mesma condição: NOT (*step0*.X) e pôde verificar-se que o programa ficava para sempre no *step0*, nunca ultrapassando esta condição. Para verificar o funcionamento da *step feature* relativa ao tempo, substituiu-se a transição que se seguia ao *step0* pela transição com a seguinte condição: *step0*.T > T#5s. Para a verificação do correcto funcionamento desta condição, utilizou-se um relógio e verificou-se que o programa apenas passava do step0 para o *step1* 5 segundos depois do step0 estar activo, tal como era esperado.

2. *Transitions*:
   a. Transição sempre verdadeira com 1 e com TRUE
   b. Transição sempre falsa com 0 e com FALSE

Descrição detalhada dos testes Sequence Flow Chart.2 – Criou-se um programa com a linguagem SFC com 2 etapas (step0 e step1) e uma transição entre eles. Nesta transição escreveu-se os valores 1, TRUE, 0 e FALSE, compilando-se o programa para cada uma destas condições. Tal como pretendido, nos valores correspondentes à primeira alínea a transição foi imediatamente ultrapassada enquanto que nos valores correspondentes à segunda alínea a transição nunca foi ultrapassada, ficando o programa preso na mesma até ser desligado.

3. Criação de uma *Action*.
   a. ST
   b. FBD
   c. SFC

Descrição detalhada dos testes Sequence Flow Chart.3 – Para o programa SFC criado, criou-se uma *action*, que foi posteriormente associada a um *Action Block*, que se encontrava ligado a uma etapa. A acção foi definida com as três diferentes linguagens: ST, FBD e SFC. De seguida, compilou-se o programa e verificou-se o funcionamento da acção através da visualização do comportamento das variáveis associadas à mesma no *debugger* do *Beremiz*.

4. Utilização de diferentes qualificadores de *actions*:
   a. N (Non-stored)
   b. R (overriding Reset)
   c. S (Set (Stored))
   d. L (time Limited)

e. D (time Delayed)
f. P (Pulse)
g. DS (Delayed and Stored)
h. SL (Stored and time Limited)

Descrição detalhada dos testes Sequence Flow Chart.4 – Acedeu-se ao livro [4] para se verificar qual o comportamento da acção associada a cada qualificador. Para isso, criou-se um programa SFC no qual se criaram 4 etapas (*step0*, *step1*, *step*2 e *step3*), colocando-se uma transição entre eles, ficando o programa disposto da seguinte maneira: [*step0*]-T-[*step1*]-T-[*step2*]-T-[*step3*]. Ao *step1* ligou-se um *action block* com uma acção associada. Para essa acção trocou-se o qualificador para os qualificadores indicados nas alíneas a até h, compilando-se o programa de seguida. Não se ligou qualquer outro *action block* às restantes etapas. Criou-se mais do que uma etapa pois alguns qualificadores fazem com que as alterações que a acção prevê deixem de se verificar assim que a etapa à qual a acção está associada deixe de estar activa. Por exemplo: se uma variável está a FALSE e a acção a coloca a TRUE, há qualificadores que fazem com que esta variável fique com o valor TRUE até que outra acção a coloque com o valor FALSE e há qualificadores em que a variável apenas terá o valor TRUE enquanto a etapa à qual o *action block* está associado estiver activa. A transição entre o step0 e o step1 verificava-se assim que uma variável externa ficasse activa, funcionando esta como um botão. Entre o *step1* e o *step2* a transição verificava-se depois de 2 segundos e entre o *setp2* e o *setp3* a transição verificava-se depois de 5 segundos. O comportamento da variável cuja acção alterava o estado foi verificado na ferramenta *debbuger* do *Beremiz*.

5. *Sequence evolution*:
   a. Colocar 2 etapas seguidas num programa SFC sem nenhuma transição entre eles.
   b. Colocar uma etapa seguida de um *jump* num programa SFC sem nenhuma transição entre eles.
   c. Colocar 2 transições seguidas.
   d. Utilização de 2 etapas ligadas directamente a uma *selection convergence*. sem nenhuma transição entre a etapa e a selection *convergence*.
   e. Utilização de 2 etapas em que cada uma a uma estava ligada a uma transição, sendo que se ligou cada uma dessas transições a uma entrada da *selection convergence*.
   f. Ligação de uma *selection convergence* a uma etapa.
   g. Ligação de uma *selection convergence* a uma transição.

h. Utilização de 2 etapas ligadas directamente a uma *simultaneous convergence*, sem nenhuma transição entre as etapas e a *simultaneous convergence*.
i. Utilização de 2 etapas em que cada uma se encontrava ligada a uma transição, sendo que se ligou cada uma dessas transições a uma entrada da *simultaneous convergence*.
j. Ligação de uma *simultaneous convergence* a uma etapa.
k. Ligação de uma *simultaneous convergence* a uma transição.
l. Ligação de uma etapa a uma *selection divergence*.
m. Ligação de uma transição a uma *selection divergence*.
n. Ligação de uma *selection divergence* a 2 etapas (cada saída ligada a uma etapa) directamente, sem nenhuma transição entre eles.
o. Ligação de uma *selection divergence* a 2 transições (cada saída ligada a uma transição).
p. Ligação de uma etapa a uma *simultaneous divergence*.
q. Ligação de uma transição a uma *simultaneous divergence*.
r. Ligação de de uma *simultaneous divergence* a 2 etapas (cada saída ligada a uma etapa) directamente, sem nenhuma transição entre eles.
s. Ligação de de uma *simultaneous divergence* a 2 transições (cada saída ligada a uma transição).
t. Tentativa de acabar o programa numa transição.

Descrição detalhada dos testes Sequence Flow Chart.5 – Os testes SFC.5 já estão explicados em cada alínea dos mesmos. Assim, criou-se um programa SFC e no mesmo realizaram-se os testes descritos nas alíneas, compilando-se o programa de seguida.

Program Organization Unit

1. Criação de um programa ST.

2. Criação de um Function Block.

3. Criação de um programa FBD.

4. Criação de um programa SFC.

5. Integração de programas em diferentes linguagens (programas de diferentes linguagens que interagem e correm em paralelo).

Descrição detalhada dos testes Program Organization Unit – Criaram-se diferentes POU's do tipo programa com as linguagens ST, FBD e SFC em que em cada um deles controlava um dos tapetes da LPF utilizada para a realização dos programas de validação referido no capítulo 5. Ou seja, um programa em ST que verificava se o tapete ST1 estava livre, mandando uma peça do tapete AT1, caso o seu sensor estivesse activo, para o ST1, se essa condição se verificasse. O programa em SFC verificava se o tapete ST2 estava livre e, caso estivesse, enviava a peça contida no tapete ST1 para o ST2. O programa em FBD continha um FB que verificava se o tapete ST3 estava livre, fazendo com que a peça que estava no tapete ST2 passasse para o tapete ST3 se não houvesse lá nenhuma peça. De seguida, criou-se uma tarefa cíclica com um período de 10 milisegundos à qual se associaram instâncias dos 3 programas criados para que funcionassem os 3 em paralelo. De seguida compilou-se o projecto e verificou-se o seu correcto funcionamento no simulador da LPF.

Comunicação

1. Adição do *plugin Modbus* ao *software*.

2. Criação de uma ligação *Modbus* e definição do modo como TCP e do funcionamento como Master.

3. Adição de um nó com o respectivo ip e porta.

4. Criação e utilização de um *request* do tipo "02-Read Discrete Inputs" e definição do número de canais e do endereço inicial.

5. Criação e utilização de um *request* do tipo "15-Write Multiple Coils" e definição do número de canais e do endereço inicial.

6. Criação de uma ligação *Modbus* e definição do modo como TCP e do funcionamento como *Slave*.

7. Adição de 2 ligações *Modbus*.

Descrição detalhada dos testes Comunicação – Cada ponto destes testes explica o que foi feito, não sendo necessária qualquer informação adicional. Depois da criação do cliente *Modbus* leu-se e escreveu-se as suas entradas através da criação de POU's, tal como descrito nos testes Program Organization Units de maneira a verificar se a comunicação estava a ser bem feita, não se limitando ao resultado da compilação do projecto.

Configurações

1. Criação de *Tasks* perióricas e não-periódicas.

2. Criação de instâncias de programas.

Descrição detalhada dos testes Recursos – Este teste está directamente associado aos testes Program Organization Unit, estando descrito na descrição detalhada dos mesmos.

5.3 Relatorio de bugs do Beremiz

O presente subcapítulo é uma extensão do capítulo anterior e lista todos os *bugs* encontrados no *software Beremiz* após a realização exaustiva dos testes a todos os seus recursos.

Enumeração de bugs

Variáveis

- Variáveis.1.u – O tipo WSTRING não é suportado pelo *software*.
- Variáveis.3.a – Erro de compilação ao adicionar o caracter '$L' (LF – *Line Feed*).
- Variáveis.3.b – O tipo *Double-byte Character Strings* não é suportado pelo *software*.
- Variáveis.6.e – Na variável do tipo *array* não é possível especificar qual o endereço dos elementos presentes em cada posição no *array*. Desta forma, é impossível conectar o *array* criado a alguma variável, fazendo com que a comunicação de variáveis do tipo *array* com dispositivos externos seja impossível neste *software*. Isto vai contra o que diz na norma que refere que deve ser possível dar um endereço a qualquer elemento da variável [5].

Funções

- Funções.2.g – O Function Block CTUD (Up-Down Counter) presente no *software Beremiz* não está definido conforme a norma IEC61131-3. Segundo a norma, o Function Block tem 5 entradas e 3 saídas, enquanto que no Beremiz, o Function Block tem 5 entradas e 5 saídas, tal como as imagens 41 e 42 ilustram.

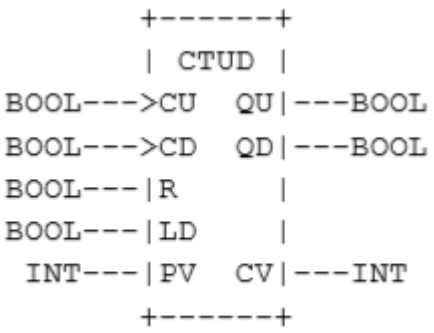

Figura 34 - Function Block Up-Down Counter na norma IEC61131-3 [5]

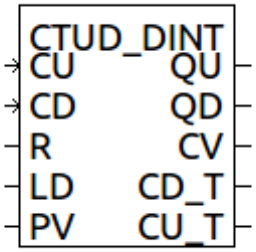

Figura 35- Function Block Up-Down Counter no Beremiz

Sequence Flow Chart

- Sequence Flow Chart.3.c – Ao contrário do que está definido na norma IEC61131-3, no programa *Beremiz* não é possível adicionar uma acção em linguagem SFC, sendo apenas possível adicionar acções ST e FBD.
- Sequence Flow Chart.5.a – Deveria ser impossível colocar 2 etapas seguidas num programa SFC sem nenhuma transição entre eles. No *Beremiz* é possível fazer isto, não dando nenhum erro de compilação.
- Sequence Flow Chart.5.b – Ao utilizar um *step* seguido de um *jump* num programa SFC sem nenhuma transição entre eles, o Beremiz compila sem qualquer problema, o que não deveria suceder.
- Sequence Flow Chart.5.d – A utilização de 2 etapas ligadas directamente a uma *selection convergence*, sem nenhuma transição entre a etapa e a *selection convergence* é possível no *Beremiz*. Ao "montar" os elementos desta forma, o *Beremiz* não dá qualquer erro de compilação.
- Sequence Flow Chart.5.g – Ao contrário do que seria expectável, é possível fazer a ligação de uma *selection convergence* a uma transição no *Beremiz,* já que ao fazer isto esta ferramenta compila sem nenhum problema.

- Sequence Flow Chart.5.j – Não deveria ser possível fazer a ligação de uma *simultaneous convergence* a uma etapa. Ainda assim, o *Beremiz* não dá qualquer erro de compilação se o utilizador fizer essa ligação.
- Sequence Flow Chart.5.n – O Beremiz compila sem qualquer problema a ligação de uma *selection divergence* a 2 etapas (cada saída ligada a uma etapa) directamente, sem nenhuma transição entre elas, o quer não deveria suceder.
- Sequence Flow Chart.5.p – A ligação de uma etapa a uma *simultaneoes divergence* não dá nenhum erro e, conforme o que está definido na norma IEC61131-3, deveria dar.
- Sequence Flow Chart.5.s – Ao se ligar uma *simultaneous divergence* a duas transições, o *Beremiz* comporta-se ao contrário do esperado. Não dá nenhum erro de compilação quando o que deveria acontecer seria a não compilação e consequente erro.

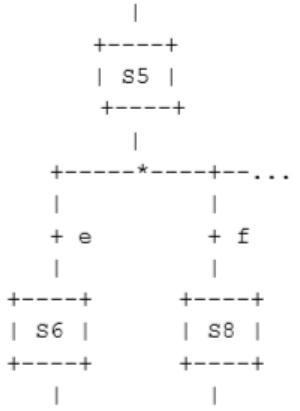

Figura 36 - Divergence of Sequence Selection [5]

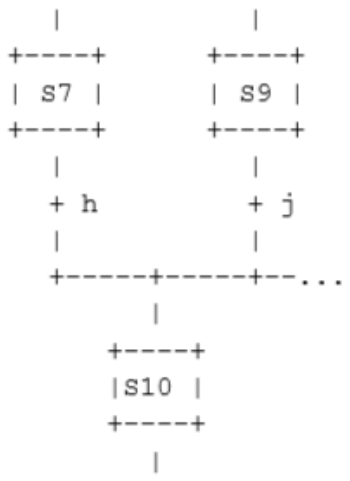

Figura 37 - Convergence of Sequence Selection [5]

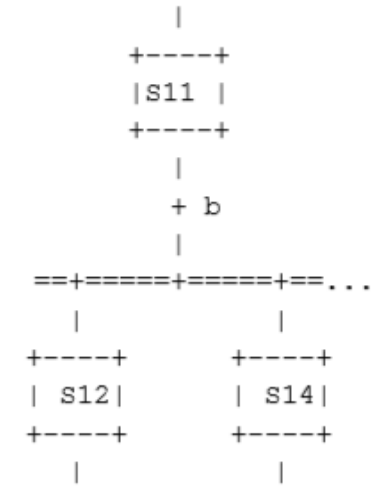

Figura 38 - Divergence of Simultaneous Sequence [5]

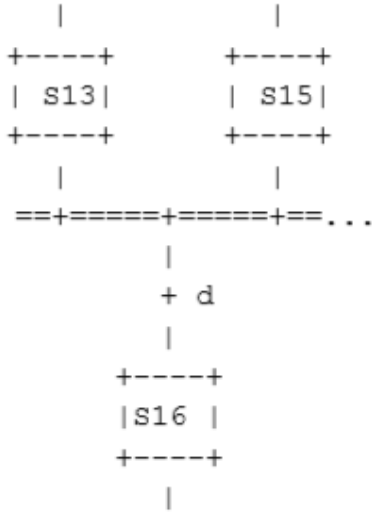

Figura 39 - Convergence of Simultaneous Sequence [5]

Comunicação

- Comunicação.6 – O protocolo *Modbus* ainda se encontra em desenvolvimento. Assim, não é ainda possível a criação de um *client Modbus slave*. Desta forma, não é possível exportar variáveis deste software através deste protocolo de comunicação. Uma vez que o protocolo *Modbus* ainda não está totalmente implementado, não é possível utilizar todas as suas funções. Estão apenas implementadas as seguintes funções :
  - 01 – Read Coils
  - 02 – Read Input Discretes
  - 03 – Read Holding Registers
  - 04 – Read Input Registers
  - 15 – Write Multiple Coils

- 16 – Write Multiple Registers
  (a 05 e a 06 não estão disponíveis)

Outros

- É necessário reiniciar o Beremiz diversas vezes devido a erros que ocorrem quando o utilizador tenta compilar o programa. Não são necessariamente erros de compilação, muitas vezes são apenas erros de interface gráfica ou erros que ocorrem sem explicação, obrigando o utilizador a reiniciar o programa para o poder continuar a utilizar.
- A interface gráfica para a atribuição de endereços *Modbus* a variáveis não está a funcionar, tendo o utilizador de realizar este processo manualmente.

5.4 Programa SCADA

Das três ferramentas testadas para a crição de programas SCADA, optou-se pela utilização do ScadaBR. Para o teste e validação do mesmo, elaborou-se um programa SCADA capaz de monitorizar a LPF, que correrá o programa de controlo especificado no capítulo 5.1. Desta forma, ambas as ferramentas *open source* estarão a controlar o mesmo objecto (o simulador), também este *open source*.

5.4.1 Comunicação

Para a comunicação entre o programa SCADA e o simulador ser possível é necessário definir-se através de que protocolo é que o ScadaBR vai ler as variáveis que se pretende monitorizar. Entre os vários protocolos de comunicação que esta ferramenta disponibiliza, escolheu-se novamente o protocolo *Modbus TCP/IP* como *data source*, cujo *Host* é o *localhost* (127.0.0.1) e a porta é a 5502, que são as definições do *slave* (o simualdor da LFP). Tal como na aplicação *Beremiz*, o tempo de actualização definido foi de 10 ms e o *timeout* 2s.

Concluída a edição das propriedades da ligação Modbus, adicionaram-se as variáveis (Figura 40). Para a adição das mesmas é apenas necessário definir o seu tipo de dados (*discrete input*, *discrete output*, *input register*, *ouput register*) e o seu *offset* (para determinar qual o seu endereço). Assim, adicionaram-se 95 *data points* do tipo *discrete input* (*status de entrada* no ScadaBR), correspondentes aos sensores da LFP e 126 *data points* do tipo *output register* (*status do coil* no ScadaBR), correspondentes aos motores da LFP.

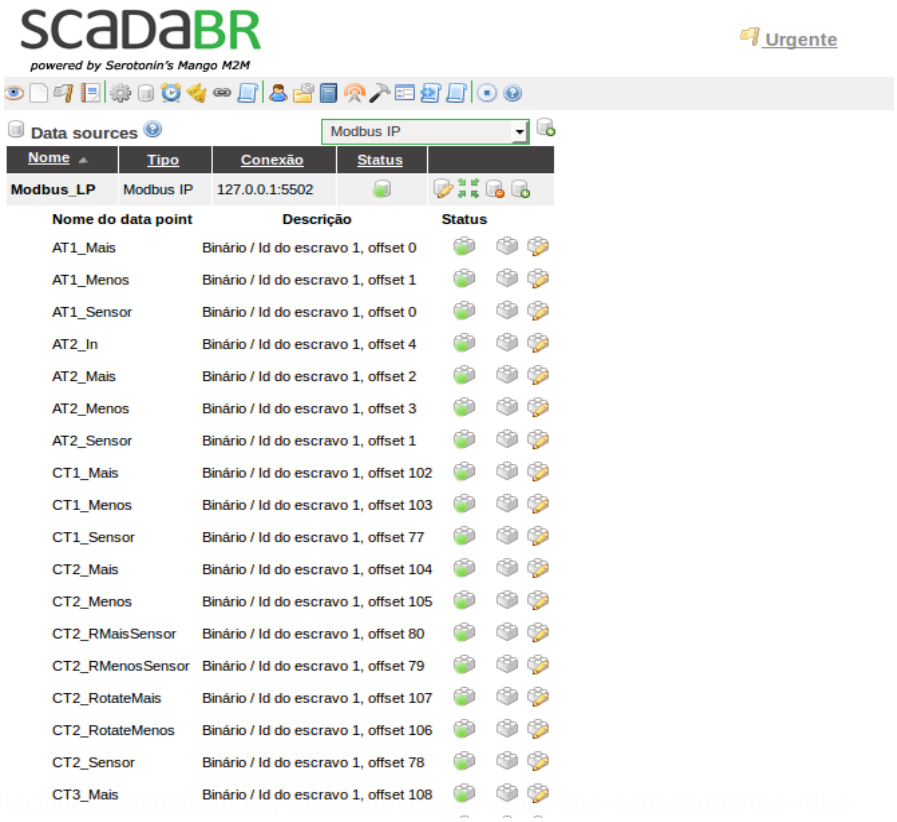

Figura 40 - Data source e data points

5.4.2 Watch List

De maneira a supervisionar os *data points* em tempo real, é possível adicioná-los à *watch list* (Figura 41), onde se pode ver qual o valor actual de cada variável bem como um gráfico de tendências instantâneas com todas as variáveis presentes na *watch list*. É ainda possível consultar cada variável em particular, podendo aceder-se ao histórico da mesma, onde se pode visualizar quais os valores que esta variável teve num determinado espaço de tempo, através de uma tabela de dados históricos ou de um gráfico à escolha do utilizador.

Figura 41 - Watch List

5.4.3 Alarmes e eventos

Ao aceder-se às propriedades de cada variável em particular, é também possível definir alarmes que ocorrem quando determinados eventos os despoletam. Assim, de maneira a melhor supervisionar a LPF criaram-se vários alarmes (ver Figura 42). Todos os sensores que estejam activos durante 10 segundos ou mais geram um alarme que informa que determinado sensor está activo há mais tempo que o normal de maneira a se poder ir verificar se está alguma peça presa no percurso. Também os motores, se estiverem a funcionar por mais de 10 segundos seguidos, fazem soar um alarme que avisa para o estranho funcionamento da LPF, permitindo a rápida correcção do problema existente. Adicionaram-se também alarmes que são accionados caso os tapetes rotativos estejam na sua posição vertical por mais de 10 segundos, uma vez que a sua posição pré-definida é a posição horizontal. Por fim, adicionaram-se alarmes caso os sensores dos tapetes CT3 ou CT8 (ver Figura 23) pois, caso as peças cheguem a estes tapetes, têm que ser retiradas já que chegando a estes tapetes as peças não têm possibilidade de chegar ao seu destino, o armazém.

É possível definir a categoria do alarme, havendo categorias com maior e menor prioridade. Quando se recebe um alarme, fica um aviso a piscar no ecrã e, carregando nele, é possível ir à lista de alarmes (Figura 43). Nesta lista é possível fazer *acknowledge* dos mesmos.

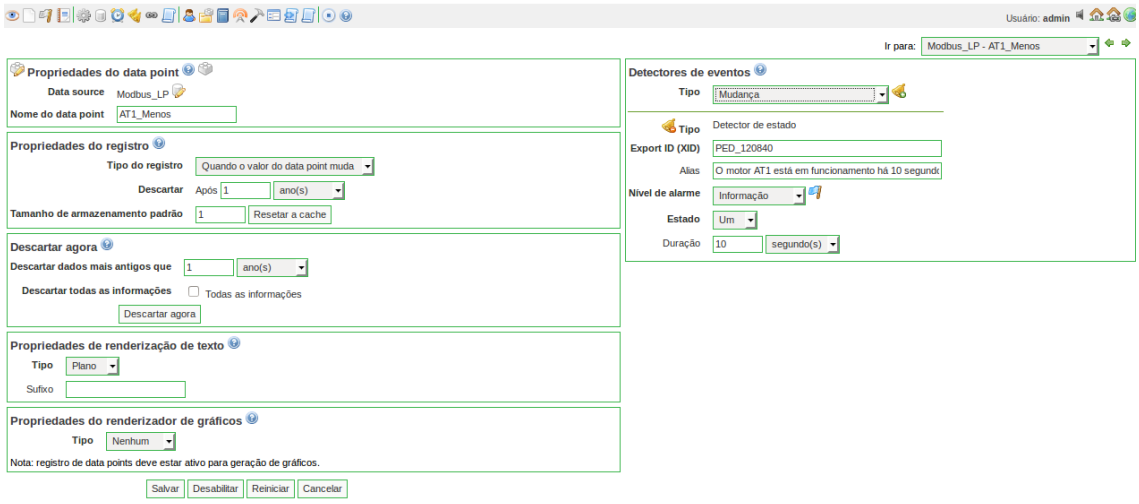

Figura 42 - Novo evento

Figura 43 - Lista de alarmes

5.4.4 Representação gráfica

Para facilitar a monitorização, criou-se uma representação gráfica (Figura 44) pois é bastante mais intuitivo supervisionar o funcionamento da LPF numa representação gráfica do que na *watch list*, onde a lista de variáveis é muito extensa.

Assim, criou-se uma interface gráfica na qual se utilizou uma imagem do simulador como fundo. Aí adicionaram-se animações próprias para a representação das variáveis booleanas, os GIF's binários. Estas animações têm um comportamento quando a variável tem o valor TRUE e outro comportamento quando tem o valor FALSE. Para os sensores, adicionaram imagens semelhantes a LED's, que estarão ligados quando o sensor tem uma peça e desligados quando o sensor nao tem nenhuma peça.

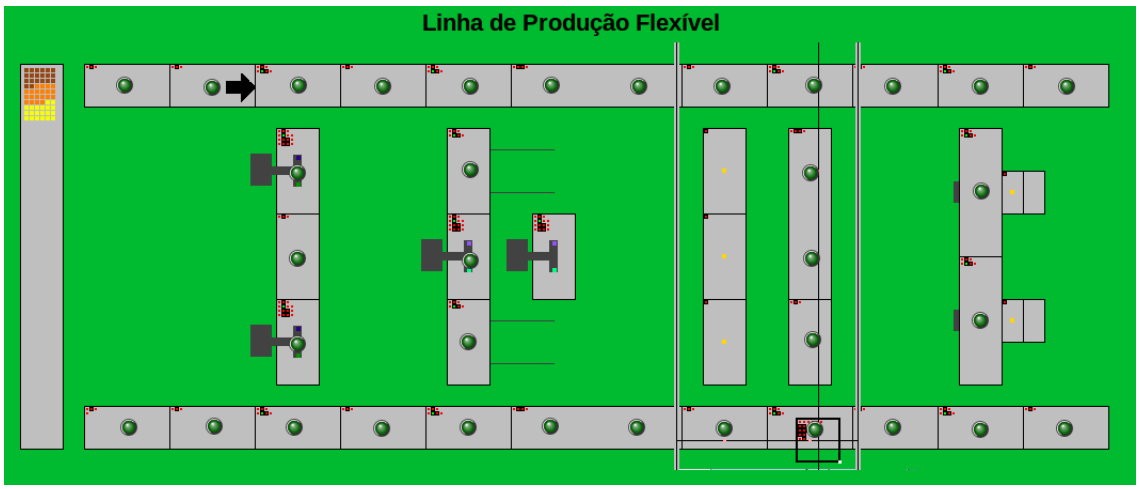

Figura 44 - Representação gráfica criada no ScadaBR

Para a supervisão dos motores, não estava disponível nenhuma imagem onde fosse claro se o motor estava parado, a andar para a esquerda, a andar para a direita ou em rotação. Assim, uma vez que a licença do programa ScadaBR é GPL, portanto de código aberto, decidiu-se acrescentar animações à biblioteca de animações já existentes. Assim, criaram-se

imagens GIF que piscam de meio em meio segundo. Foram criadas 6 imagens, 2 correspondentes aos movimentos lineares (para a esquerda e para a direita) e 4 correspondentes às rotações.

Figura 45 - Imagens gif adicionadas

Para que estas imagens fiquem disponíveis no programa ScadaBR, foi necessário criar uma pasta com o nome pretendido (neste caso o nome escolhido foi *Arrows*) dentro da pasta *graphics* que se encontra dentro do ficheiro ScadaBR.war que contém a aplicação ScadaBR para o SO Linux. Dentro da pasta *Arrows*, para além de ser necessário colocar lá as imagens criadas, é também necessário colocar lá um ficheiro com o nome info.txt. O conteúdo deste documento de texto tem que ser o seguinte:

name=Arrows
type=imageSet
#width=32
#height=32
text.x=5
text.y=32
(o utilizador pode definir os valores que quiser em frente ao "=")

5.4.5 Relatório

Para concluir a configuração do SCADA, criou-se um relatório que contém informação sobre todos os eventos despoletados por qualquer variável. Esse relatório é enviado para o e-mail ee08282@fe.up.pt.

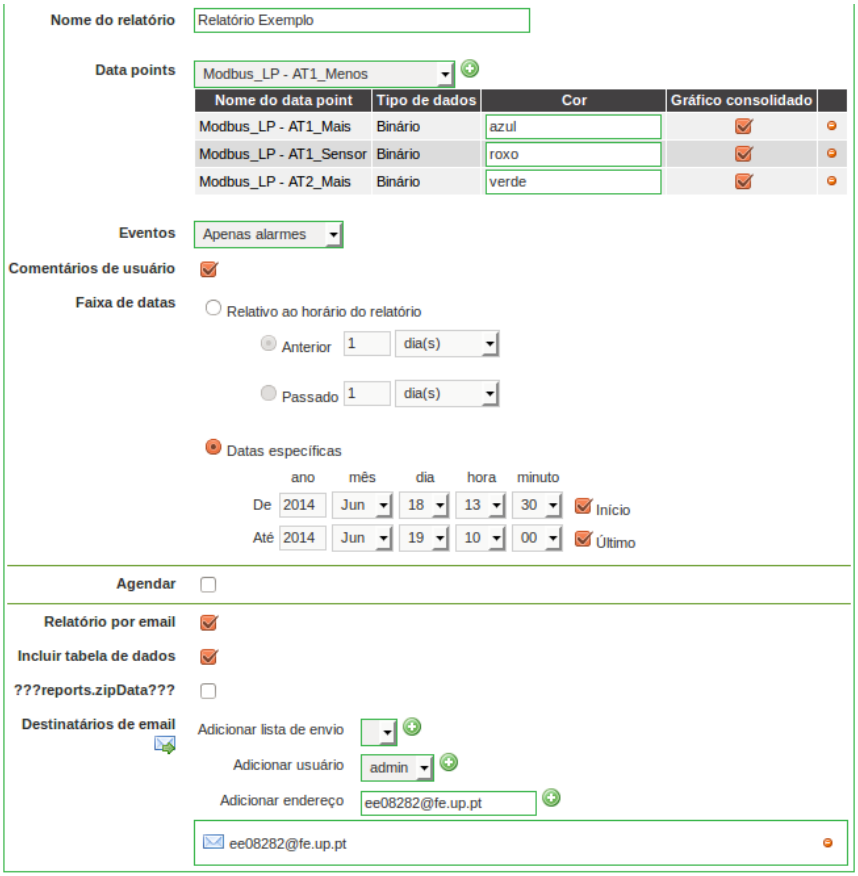

Figura 46 - Novo Relatório

5.5 Testes realizados ao software ScadaBR

Apesar de haverem já inúmeros projectos feitos através da utilização do ScadaBR, realizaram-se alguns testes onde o principal objectivo era a exploração das potencialidades deste software e perceber o funcionamento do mesmo. Os testes não foram feitos com o intuito de encontrar *bugs*, que seriam mesmo assim reportados caso se descubrissem. Com estes testes pretende-se também comprovar que o ScadaBR funciona de acordo com a arquitectura típica de um SCADA e se oferece as suas funcionalidades mais comuns como a capaciade de fazer a aquisição de variáveis de equipamentos externos através de diferentes protocolos e fazer a sua supervisão em tempo real, criar interfaces HMI, guardar histórico de variáveis, disparar alarmes e gerar relatórios.

Aquisição de dados e variáveis

1. Criação de um *data source* com o protocolo *Modbus IP*:
2. Criação de um *data source* virtual.
3. Criação de um *meta data source*.

4. Criação de *data points* do com o tipo de dados:
    a. Binário
    b. Numérico
    c. Alfanumérico
    d. Multiestados

Descrição detalhada do teste Aquisição de dados e variáveis - Para a aquisição de dados, testou-se o protocolo *Modbus* TCP/IP onde se fez a comunicação com o simulador da LPF. Para esse efeito, definiu-se o Host como *localhost* e porta 5502 e utilizou-se o período de actualização de 10ms e o timeout de 2s. Uma vez que o simulador apenas tem variáveis booleanas, adicionaram-se mais duas ligações: uma *data source virtual* e uma *meta data source*, para ambas é apenas necessário definir o período de actualização que foi também de 10ms. Relativamente às variáveis binárias da ligação Modbus, adicionaram-se dois *data points* com tipos de dados diferentes: status de entrada (correspondente ao *read input register*) para os sensores e status de *coil* (correspondente ao *write coils*) para os actuadores. Para estas variáveis é também necessário definir qual o seu *offset* de maneira a definir qual o endereço de cada variável. Verificou-se o funcionamento do *data source Modbus* e dos seus respectivos *data points* através da visualização do seu valor, observado em tempo real na *watch list*, ao mesmo tempo que se forçava o valor dessas mesmas variáveis directamente no simulador. O *data source virtual* foi criado de maneira a se poder verificar o funcionamento dos outros tipos de variáveis que o ScadaBR tem disponíveis. Para esta ligação, criaram-se variáveis do tipo numérico para as quais se definiu o tipo de alteração browniano que significa que a variável vai variar aleatoriamente entre um valor mínimo e máximo que tiveram que ser definidos. Uma vez que a *data source* é virtual, não se ligando a nenhum dispositivo físico real, os valores das suas variáveis são simulados. Adicionaram-se também variáveis multiestados para as quais se tem que definir quais os estados entre os quais esta variável vai variar. Por fim, adicionaram-se variáveis alfanuméricas para as quais se deve definir um valor inicial. Explorando os *data points*, verifica-se que é possível ainda editar as suas propriedades, onde se pode definir: o período de tempo que se deseja gravar no histórico de dados, qual o gráfico que se pretende utilizar para se ver o comportamento da variável e ainda definir uma unidade para essa variável (no caso das variáveis numéricas).

Eventos and Alarmes

1. Criação de um eventos
    a. Mudança
    b. Estado
    c. Limite superior
    d. Limite inferior

Descrição detalhada dos testes Eventos e Alarmes.1 – Para se criarem alarmes, é necessário que estes estejam associados a um determinado evento que pode ser definido nas propriedades de cada *data point*. Quando o utilizador cria um evento, pode escolher entre um evento de mudança, de estado, de alteração de estado, limite superior e limite inferior. Para estes eventos o utilizador pode definir um nível de alarme ou não. Os níveis de alarme disponíveis são: nenhum alarme, informação, urgente, crítico e *life safety*. Para além dos alarmes definidos pelo utilizador, existem ainda alarmes do sistema, cujo nível pode ser alterado nas definições de sistema do ScadaBR. Para um evento do tipo mudança, é apenas necessário definir o *alias* que é a frase de alerta que vai aparecer na lista de alarmes caso este alarme dispare. Este evento ficará activo cada vez que a variável mude de estado (neste caso, de 1 para 0 ou de 0 para 1). Testou-se este evento num *data point* binário. O evento de estado foi também testado para o *data point* binário, sendo necessário definir o *alias*, a duração e o estado. Por exemplo: estado=1, alias="erro" e duração=10s; Para este caso, o evento ficará activo caso a variável binária esteja com o valor TRUE durante 10 ou mais segundos. Para os eventos de limite inferior e superior é necessário definir esse mesmo limite e um tempo de duração, bem como o alias. Caso a variável este abaixo ou acima desse valor, respectivamente, durante mais tempo do que o tempo de duração disponível, esse evento ficará activo, fazendo disparar um alarme caso o utilizador assim o defina. Para o teste deste tipo de evento utilizou-se uma variável numérica de um *data source virtual*. Para todos os testes verificou-se se os tempos de duração estabelecidos pelo utilizador se verificavam.

Interface HMI

1. Escolha de imagens de fundo com diferentes resoluções.

2. Utilização de elementos gráficos
    a. GIF binário
    b. GIF analógico
    c. GIF dinâmico
    d. GIF multiestados
    e. HTML
    f. Gráficos
    g. Botão
    h. Lista de alarmes

Descrição detalhada dos testes Interface HMI.2 – Para a criação de uma interface HMI, o utilizador apenas tem que aceder a este menu através do atalho existente na barra de tarefas do ScadaBR. De seguida, é apenas necessário definir um nome e uma imagem devendo o utilizador adicionar *widgets* (elementos gráficos) à imagem de fundo que definiu que estão disponíveis para escolha num menu e podem ser arrastado através da funcionalidade *drag and drop*. Quanto aos GIF's, estes não estão disponíveis para qualquer *data point*, isto é, cada GIF dá para um tipo de data point específico. Por exemplo, o GIF binário só dá para variáveis

binárias, os GIF's analógico e dinâmico só dão para variáveis numéricas e o GIF multiestados só dá para variáveis multiestados. Para qualquer *widget* GIF é necessário fazer a associação a um *data point*, seleccionando-o num menu *drill* nas opções do elemento gráfico. Nos GIF's analógicos é necessário definir um mínimo e um máximo. Para este tipo de GIF's normalmente utilizam-se tanques ou recipientes que se encherão e esvaziarão conforme o valor da variável esteja situado na escala entre o mínimo e o máximo definidos pelo utilizador. Os GIF's dinâmicos são muito semelhantes aos GIF's analógicos só que em vez do utilizador escolher uma imagem de uma biblioteca, escolhe uma animação que pode ser uma barra horizontal, uma barra vertical ou um mostrador redondo. Para os GIF's multiestados é apenas necessário associar uma imagem da biblioteca a cada estado da variável multiestados. Por fim, para os GIF's binários é necessário definir qual a imagem que aparecerá quando o valor da variável é 0 e quando o valor da variável é 1. Todas as definições relativas aos elementos gráficos são facilmente definidas pelo utilizador uma vez que todas as definições são escolhidas através de menus *drill*. Os elementos HTML são elementos onde o utilizador pode escrever consoante o definido na programação HTML e podem ser utilizados para se escrever um título na representação gráfica, por exemplo. A adição de gráficos aos quais se tem que associar uma variável serve para visualizar em tempo real a variação dos *data points*. Também se testou a adição de botões à representação gráfica aos quais se tem que associar uma variável binária. Ao carregar neste elemento gráfico, o utilizador consegue alterar o valor da variável de 0 para 1 e de 1 para 0. Por fim, o último elemento gráfico que se adicionou foi a lista de alarmes, podendo aceder a esta sem sair da representação gráfica.

Relatórios

1. Criação de um relatório

Descrição detalhada dos testes Relatórios – Criou-se um relatório no qual se adicionaram as variáveis binárias correspondentes aos sensores e actuadores do simulador da LPF. Relativamente a estas variáveis, definiu-se no relatório a data à qual este se referia (fez-se um relatório referente ao dia anterior e um relatório de 10 de Junho de 2014 a 20 de Junho de 2014). Neste relatório incluiram-se todos os eventos relativos a estas variáveis que estiveram activos durante a referida data. Activou-se também a opção de agendamento para o relatório ser feito de semana a semana Para concluir a elaboração do relatório, adicionou-se o e-mail ee08282@gmail.com para onde o relatório é enviado.

Scripts

1. Criação de um script que utilize apenas uma variável
2. Criação de um script que utilize duas variáveis
3. Criação de um script que utilize lógica

Descrição detalhada dos testes Scripts – A utilização dos scripts está directamente ligada à utilização de *meta data sources*. Ao adicionar-se um *data point* a este tipo de dados, é necessário escrever um script para essa variável cuja linguagem é *javascript*. De maneira a explorar melhor esta funcionalidade, criaram-se 3 variáveis numéricas numa *data source* virtual também criada para estes testes: temperatura, tensão e corrente. Para o primeiro teste, adicionou-se um data point com o nome temperaturaKelvin à *meta data source* criada anteriormente e definiu-se a variável Temperatura como context (p1 como nome da var a ser utilizada no script). Ao adicionar-se uma variável como context, o utilizador pode utilizá-la na escrita do script. O script que se escreveu para o primeiro teste é muito simples e retrata uma simples conversão de unidades. O script continha o seguinte texto: "return p1.value+273,15". Desta forma, o valor do *data point* temperaturaKelvin estará sempre dependente do valor da variável temperatura. De seguida, adicionou-se outro *data point* à *meta data source*, este com o nome potência. As variáveis de contexto deste *data point* foram a tensão (p1) e a corrente (p2) referidas anteriormente e, neste script, escreveu-se o seguinte: "return p1.value * p2.value". Para uma avaliação um bocado mais aprofundada dos scripts, criou-se um script onde se utilizam comparadores de valores como o =, > e > e expressões lógicas como o IF. Assim, criou-se mais uma variável ao *meta data source* com o nome clima do tipo alfanumérico à qual se associou como contexto a variável temperatura (p1). No script deste data point, escreveu-se o seguinte:

```
if (p221.value >= 30)
      return "quente";
else
   return "frio";
```

5.6 Considerações sobre o ScadaBR

Tal como esperado, não foi encontrado nenhum problema com os testes realizados ao ScadaBR já que este software já lançou a sua versão 1.0, versão que já se encontra devidamente testada. As únicas dificuldades encontradas foram na escrita dos scripts devido às limitações em *javascript*. Desta forma, testaram-se apenas *scripts* algo simples. A API também não foi testada pois não era uma prioridade. Ainda assim, procurou-se nos fóruns do ScadaBR relatos da sua utilização que comprovam que esta funcionalidade está também operacional [14] [73] [74].

Esta ferramenta demonstrou ser bastante completa e a sua interacção com o utilizador é de extrema facilidade para este. Além desta característica, existe bastante documentação online, bem como manuais de utilização e tutoriais para o desenvolvimento de projectos mais ou menos complicados.

Comparação com características típicas de um SCADA

Como referido no capítulo 2.2, a arquitectura de *hardware* de um sistema SCADA é normalmente constituída por sensores ou outros elementos de campo que se pretendem medir/controlar (nos testes realizados, os elementos de campo são os sensores e actuadores do simulador do LPF), um sistema de supervisão capaz de receber os valores desses elementos de campo através de diferentes protocolos, enviando-lhes comandos de controlo e uma interface HMI que permite a supervisão e controlo do processo pelo utilizador (ver Figura 1).

O ScadaBR segue a arquitectura típica do SCADA uma vez que permite adquirir variáveis através dos mais diversos protocolos como o *Modbus*, HTTP, DNP3, OpenV4J, IEC101, entre outros e dispõe de um sistema de supervisão que permite a supervisão das variáveis e o controlo das mesmas, através de uma interface HMI que tanto pode ser uma representação gráfica como a watch list, onde o utilizador pode monitorizar o valor das variáveis em tempo real através de gráficos, elementos gráficos ou tabelas de valores.

Além de seguir a arquitectura típica de um SCADA, o ScadaBR oferece também todas as características mais importantes de um SCADA que são:

- Aquisição de variáveis de diferente plataformas externas através de vários protocolos de comunicação
- Apresentar dados em tempo real para a monitorização do processo
- Fornecer interfaces gráficas que apresentam o estado actual do processo
- Gerar relatórios
- Apresentar gráficos de histórico do comportamento das variáveis
- Disparar alarmes que podem ser geridos pelo utilizador.

Devido às suas características, a avaliação desta ferramenta é portanto extremamente positiva.

Capítulo 6

Conclusões

Após se ter realizado uma extensa pesquisa sobre as ferramentas *open source* existentes no mercado, tanto para a automatização de processos como para o desenvolvimento de interfaces SCADA, e realizado testes em algumas delas, pode concluir-se que existem ferramentas *open source* capazes de entrar no mundo industrial.

Quanto à automatização de processos, o *Beremiz* provou ser uma ferramenta com muitas potencialidades, um IDE em conformidade com a norma IEC 61131-3 que é tão capaz quanto as ferramentas para a mesma finalidade cuja licença é comercial, como o ISAGRAF. Por vezes, os constantes erros que o utilizador recebe quando se encontra a fazer programas podem tornar a sua utilização maçadora, erros esses que não têm qualquer razão aparente mas que obrigam o utilizador a estar constantemente a reiniciar o *Beremiz*. Quanto a esta ferramenta, existem ainda alguns bugs, referidos no capítulo 5., que devem ser corrigidos. A sua maior limitação é a existência de poucas maneiras de comunicação, estando apenas disponíveis os plugins para a comunicação *CanOpen* e para a comunicação *Modbus* (sendo que esta funciona apenas no modo master e há duas funções que não se encontram implementadas - 05 – *write single coil* - e 06 – *wirte single register*).

No que toca a ferramentas para a criação de interfaces do tipo SCADA, foi possível constatar a existência de diversas ferramentas *open source* no mercado. Depois de uma leitura aprofundada sobre o tema, escolheram-se três ferramentas SCADA para serem testadas. A ferramenta *openSCADA* era uma das mais famosas, decidindo-se por essa razão realizar uma aplicação para a validação da mesma. Contudo, devido ao facto desta ferramenta se ter movido recentemente para a eclipse, formando o *Eclipse SCADA* fez que se encontrassem alguns obstáculos na criação de novos projectos uma vez que a migração não está ainda 100% concluída. Para além do *openSCADA*, decidiu-se também testar as ferramentas *PVBrowser* e *ScadaBR* que têm também muitos utilizadores.

Quanto ao *PVBrowser*, concluiu-se que esta ferramenta tem a capacidade de supervisionar sistemas mas necessita que o seu programador tenha conhecimentos de programação na linguagem C/C++ ou Lua, o que constitui um obstáculo para os utilizadores.

Já o *ScadaBR*, que é baseado no software *Mango* M2M, é bastante intuitivo e muito fácil de aprender a utilizar. Assim, optou-se por utilizar esta ferramenta para a realização de uma aplicação SCADA capaz de supervisionar e monitorizar a Linha de Produção Flexível controlada por uma aplicação realizada no software Beremiz, no contexto desta dissertação. O resultado desta aplicação foi bastante satisfatório, demonstrando que apesar da limitação quanto aos widgets, o ScadaBR é capaz de criar aplicações do tipo SCADA tal como as ferramentas de licença comercial existentes no mercado. A limitação quanto às animações gráficas pode ser suprimida uma vez que é fácil de adicionar novos widgets ao ScadaBR, ja que este é um software de código aberto.

6.1 Contributo

O contributo dado nesta dissertação para o tema foram os programas criados no âmbito da validação e teste das ferramentas. Para além destes programas, realizaram-se também testes para duas ferramentas para verificar o funcionamento das mesmas. No caso do *software Beremiz*, testou-se a sua conformidade com a norma IEC61131-3 tão exaustivamente quanto possível, encontrando-se alguns erros. O par testes realizados/bugs encontrados é também um contributo dado por esta dissertação.

6.2 Trabalho futuro

De maneira a continuar o trabalho desenvolvido nesta dissertação, sugere-se que o plugin Modbus para o *software Beremiz* seja acabado, faltando para isso a adição de duas *public functions* (05 e 06) e o funcionamento como *Slave* (apenas o modo Modbus Master se encontra implementado). Sugere-se também a correcção dos *bugs* encontrados no mesmo *software*.

Anexo A

Tutorial Beremiz

Abstract

This tutorial was made in order to be a small demonstration of the functionalities of Beremiz. A user who is using Beremiz for the first time will have some doubts and this tutorial will answer many questions that these users might have.

The simulator that will be controlled in this tutorial is an open-source software developed to simulate a production line.

Installation

Linux

In first place, you need to install all the software that is necessary. To take the best of Beremiz, you should download and install any version of Linux. An example is Linux Ubuntu: http://www.ubuntu.com/download/desktop.

If you are not a Linux user and you don't want to install this operating system, you can use a Virtual Machine, such as VMWare or VirtualBox, and run Linux inside of your operating system.

Python and Java

After installing Linux, you have to install python so that you can run Beremiz and you have to install Java so that you can run the simulator. You can install python and Java from command line.

- Commands to install python:

sudo apt-get install build-essential bison flex autoconf
sudo apt-get install python-wxgtk2.8 pyro mercurial
sudo apt-get install python-numpy python-nevow python-matplotlib python-lxml

- Commands to install Java:

sudo apt-get install default-jdk

Beremiz

Now everything is prepared for us to install Beremiz. You can download and install Beremiz from command line to and the commands that you have to write in the command line are the following:

mkdir ~/Beremiz
cd ~/Beremiz

hg clone http://dev.automforge.net/beremiz
hg clone http://dev.automforge.net/matiec

cd ~/Beremiz/matiec

autoreconf
./configure
make

cd ~/Beremiz
hg clone http://dev.automforge.net/CanFestival-3

cd ~/Beremiz/CanFestival-3
./configure --can=virtual
make

To run Beremiz you have to go to beremiz folder in command line and run the executable Beremiz.py as is described in this commands.

cd ~/Beremiz/beremiz
python Beremiz.py

Modbus plugin

Beremiz is now installed in your computer. To control the simulator, you have to install the Modbus protocol plugin that can be downloaded in https://feupload.fe.up.pt/get/M4K2mltjPgB9lnH.

To install this Modbus plugin for Beremiz, you should follow those instructions:

- Untar the file mb_plugin.tar.gz to the directory: "…./beremiz_folder/beremiz"
- Untar the file Modbus_lib.tar.gz to the directory: "…./beremiz_folder/"
- Compile the Modbus libraries: run 'make' inside "…./beremiz_folder/Modbus"
- Add the following line to features.py

```
catalog = [
('canfestival', _('CANopen support'), _('Map located variables over CANopen'), 'canfestival.canfestival.RootClass'),
('modbus', _('Modbus Client support'), _('Map located variables over Modbus'), 'modbus.modbus.RootClass'),
...
```

Simulator

The last thing you have to install is the simulator. You can download it by clicking in https://feupload.fe.up.pt/get/SpUsvMeIhme5Ovn. To install it, all you have to do is extract all its content to a folder. Then, to run the simulator, all you have to do is run the file sfs.jar or write the following in the command line:

cd ~/Simulator_Folder

Java -jar sfs.jar

Create a project

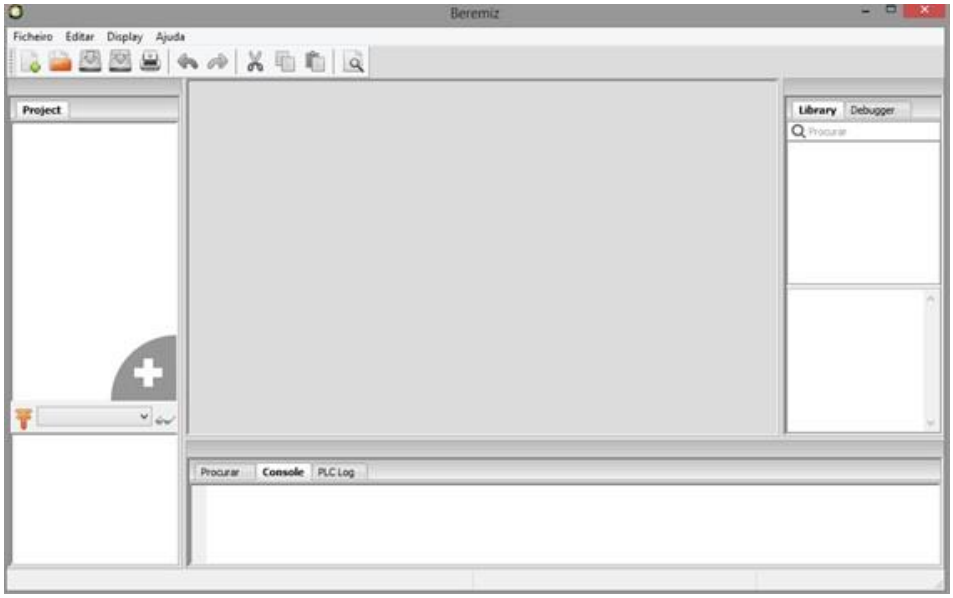

Figura 47 - Beremiz Interface

When you open Beremiz, it should look like this:

To create a new project, you should click in Ficheiro and select New (or just press CTRL+N). After that, you have to choose the location of the folder where you want to save your project. It's very intuitive.

If you want to load a created project, you should click in Ficheiro as well and select Open (or just press CTRL+N) or Recent Projects, if you have worked in that project recently.

After choosing the location of the project folder, you should now have a new project created that will appear in your project menu in Beremiz with a resource associated.

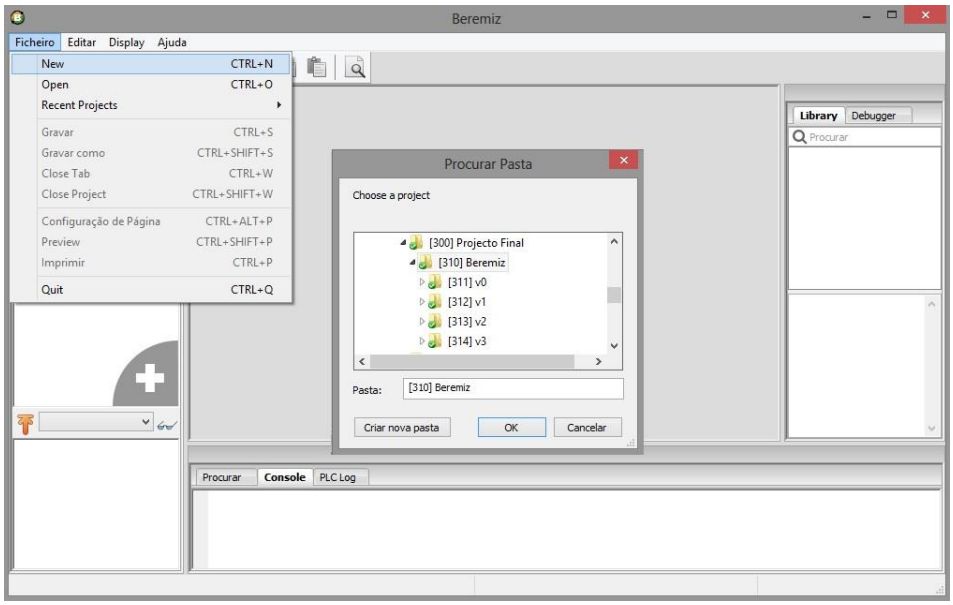

Figura 48 - New Project

Now, in order to add a program, a function block, a communication protocol or any other thing, you have to click in the + sign in your project menu.

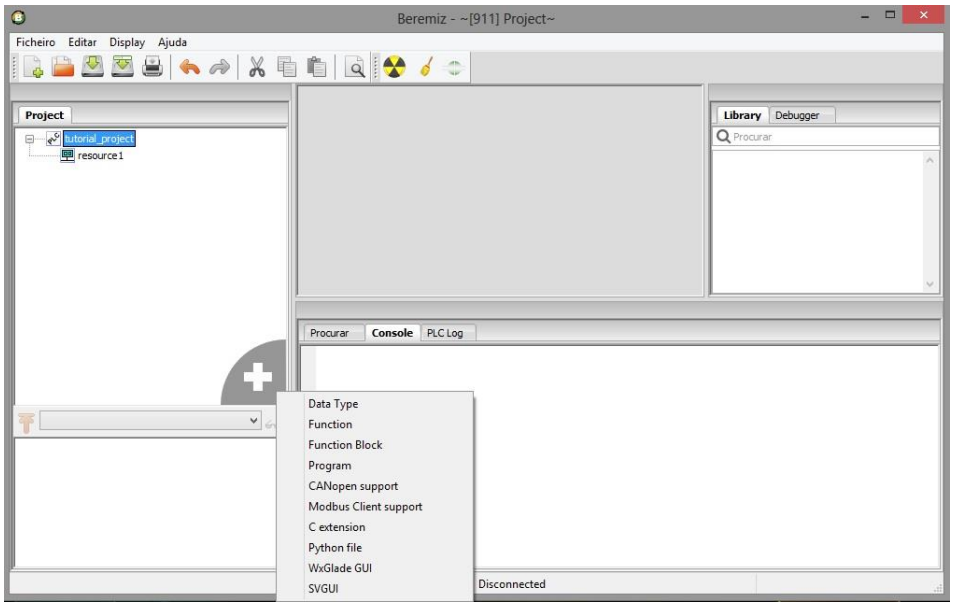

Figura 49 - Adding new features to a Beremiz Project

Configuring Communications Protocol

In order to control the simulator, you should add a new Modbus Client Support. The simulator does not work with CanOpen, that was the reason why we had to add the Modbus plugin to Beremiz. So after adding “Modbus Client Support” to the project, you have to do the following:

1. Configure the Modbus library to work as 'Master', using 'TCP'
2. Add a "ModbusIPNode" (right click in the Modbus Client created in step 1)
3. Configure the Modbus node with the IP address and port of the Modbus Slave (In this case the IP is 127.0.0.1 because we are using the localhost and the port is 5502)
4. Add a "ModbusRequest" to the Modbus Node (right click in the Modbus node)
5. Configure that "ModbusRequest"

To control the simulator, we will need two Modbus requests. One request to read the inputs and other request to write the outputs. If we check the io file that comes with the simulator, we can see that we have 95 inputs and 126 outputs. Because of that reason, we should configure your requests with those numbers, as explained in the Figures 50 and 51.

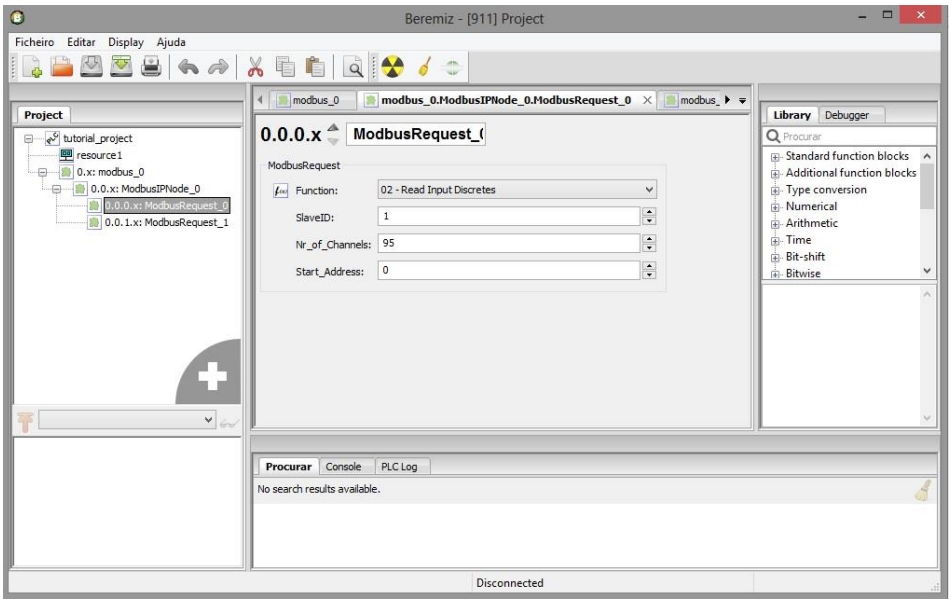

Figura 50 - Modbus Request - Read Input Discretes

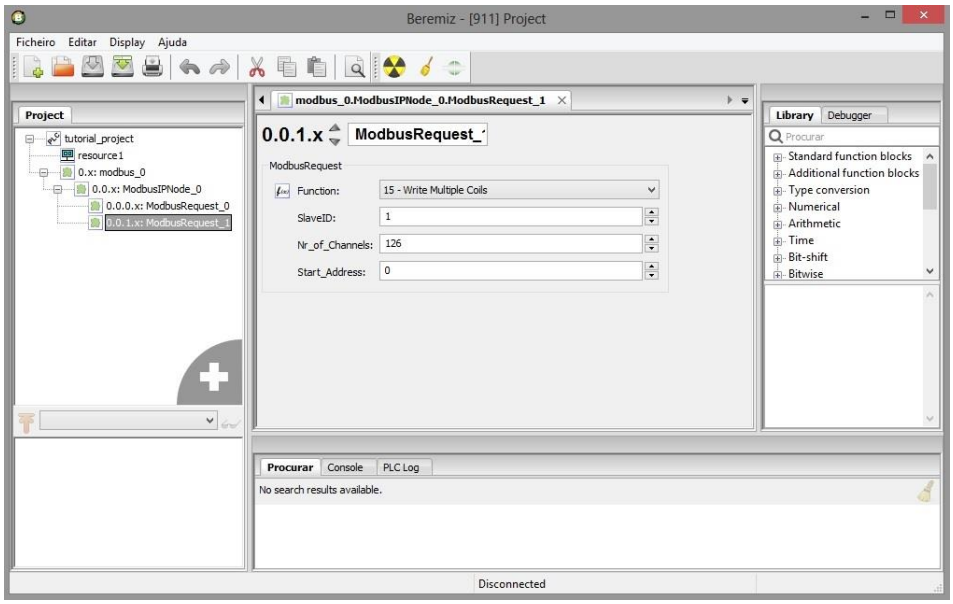

Figura 51 - Modbus Request - Write Multiple Coils

Creating a POU

To create a POU (Program Organization Unit) you should follow the same procedure that was explained in the configuration of the communications protocol. After clicking the + sign, you can choose any POU type (FunctionBlock, Function or Program). In this tutorial, will only be demonstrated the use of a program. In the next chapters, the creation of different language programs will be explained with more detail. So, after adding a new program to your project and choosing the language of that program, you need to add a task and instantiate that same program, as defined in IEC61131-3. To accomplish these steps, you should go to your “resource menu” and click at the green + sign in the task section.

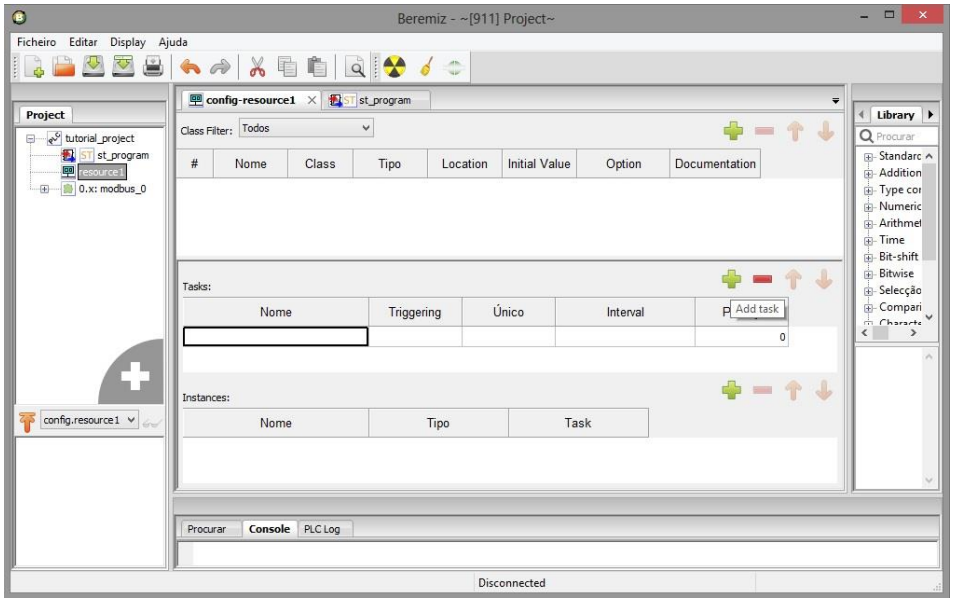

Figura 52 - New task

After adding a new task to your project, you have to configure its name, triggering and interval. The “priority” and the “unico” fields are optional and you do not have to fill them. The name does not need any explanation, you can write any name that you want. In the triggering field, you have to choose between “cyclic” and “interrupt” (in the programs that we will create in this tutorial, we will use the cyclic mode). Finally, in the interval field, you have to choose from what time to time the task will be repeated.

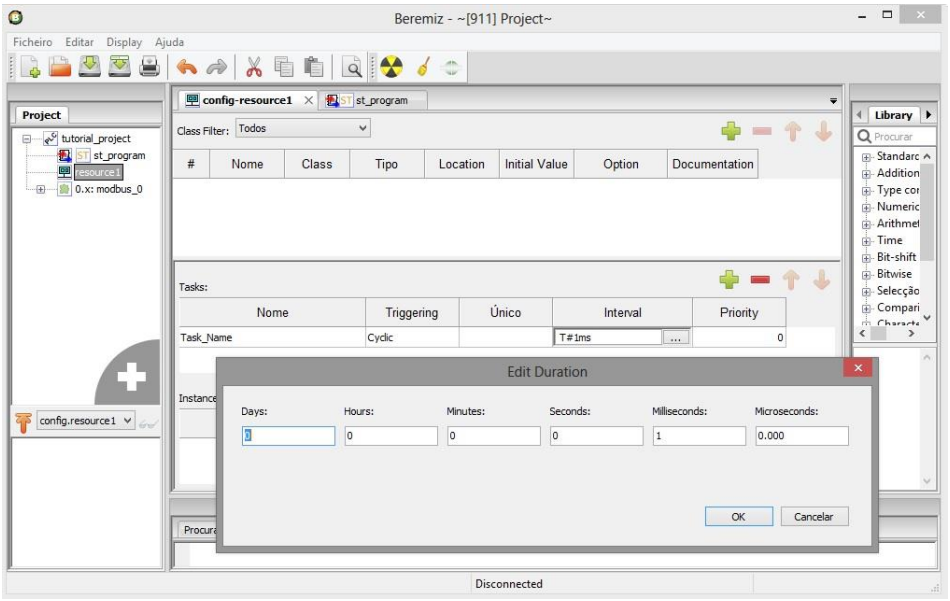

Figura 53 - Configuring a new task

Now, all that you have to do to end this step is configuring a new instance. To do that, you should do the same thing that you have done when you created a new task. You have to go the instance section and create in the green + sign. After that, you only need to choose a name to the instance and associate it to a program and to a task.

Creating variables

To write your program, you need to have variables. The creation of variables is relatively simple but you have to know the IEC61131-3 language to configure it properly. In this tutorial, the only type of variables that will be used is the BOOL type.

You have two options to add variables. You can create local variables in your resource menu and then instantiate them as external variables in your POU. This is a good option if you have to use the same variable multiple times in different POUs because that way you don’t have to write the location of the variable every time you create a new POU where that variable will be used.

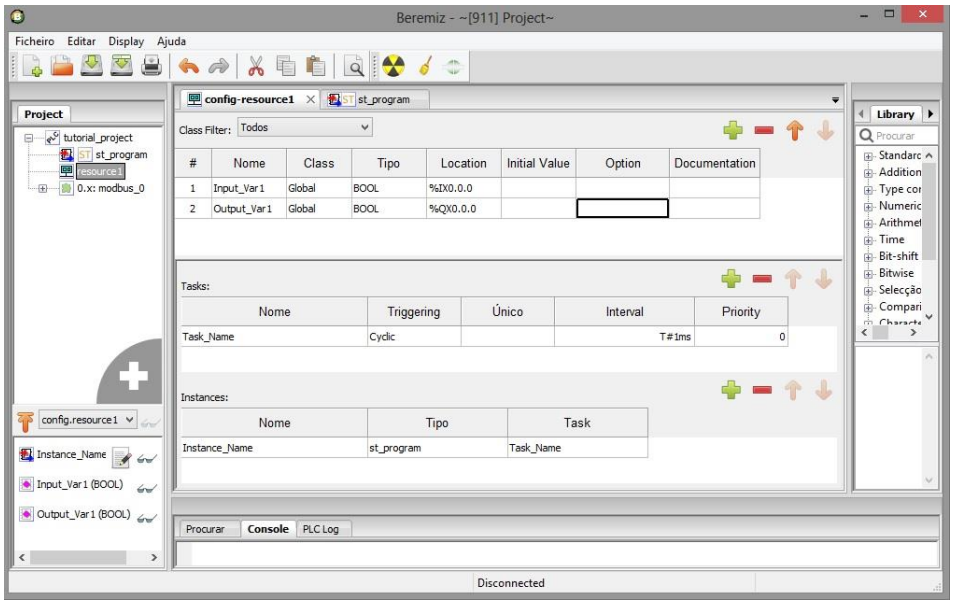

Figura 54 - Global Variables

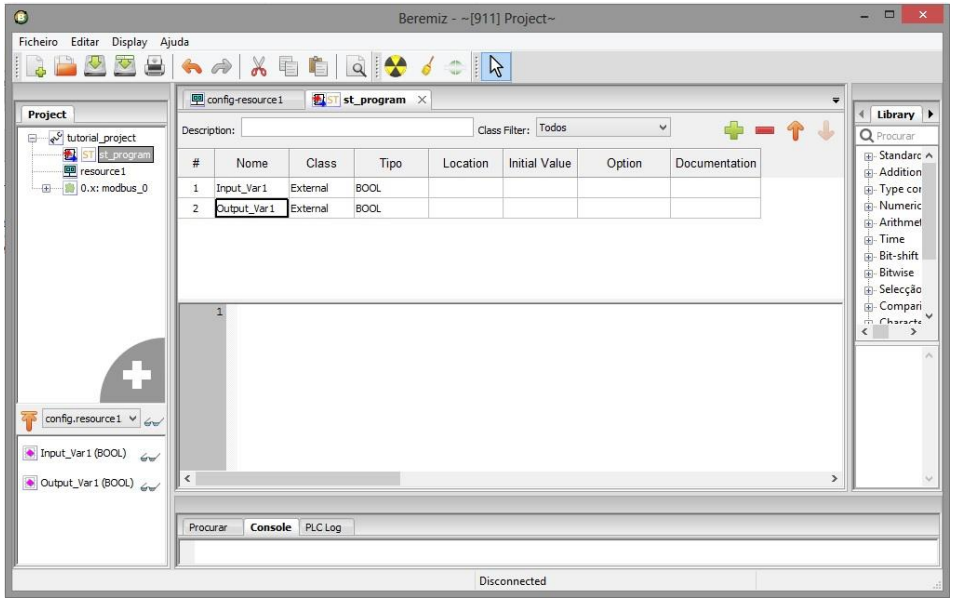

Figura 55 - Extermal Variables

Otherwise, you can create the variable as a local variable in the POU where you will use it instead of creating that variable in the resource menu.

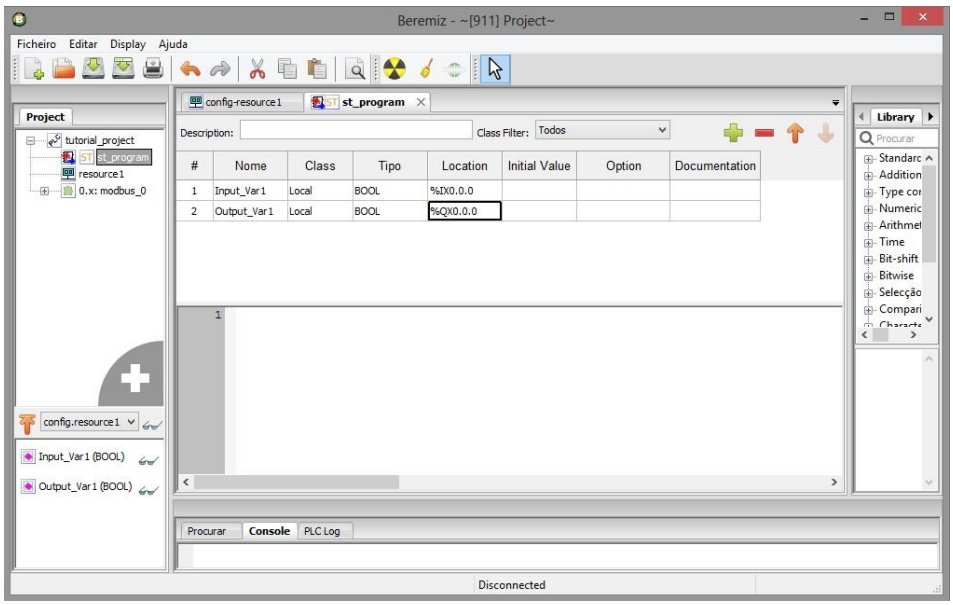

Figura 56 - Local Variables

As you can see in the Figures 54, 55 and 56, the creation of the variables is not simply clicking in the green + sign in the variables section and writing its name. To complete the creation of a variable successfully, you must choose the class, type and location of the variable. The initial value, option and documentation fields are optional and in this tutorial will not be filled because they are not necessary. In the class field, you have to decide if the variable is local, external, global, temp, etc. If you choose the class external, you have to write the same name of a global variable created previously. After giving the same name to an external variable, Beremiz understands that the external variable refers to the global variable created and you should not write the location because Beremiz already knows it.

Differently, if you define a variable as local or global, you need to write the location. Well, due to the development phase that Beremiz is in, you cannot define the location graphically. So, to write the location manually, you have to be aware to IEC61131-3. In this case, we are managing a BOOL variable so in the location we have to write %IX0.0.0 or %QX0.0.0 if we are writing an input location or an output location, respectively. If we were using a WORD variable, it would be %IW0.0.0; if we were using a double word, it would be %ID0.0.0, etc. To configure your variables properly, you have to know IEC61131-3!

This is not all! The location of your variable must be directed to the Modbus client that you added to your project. So, if you go back to the “Configuring Communications Protocol” chapter, you will remember that you have created two different requests. The first request was a “read input discretes” and it is located in 0.0.0.x and the second request is located in

0.0.1.x (x is the location of the variable). I will give you an example to be more clear to you how it should be done.

If you resort to the io.xls file which you have downloaded along with the simulator, you will see that in the first three lines you have the following content:

1,AT1,Warehouse Out,O,Motor +,0
1,AT1,Warehouse Out,O,Motor -,1
1,AT1,Warehouse Out,I,Sensor 0,0

If you pay attention to this, you can see that the first two variables are outputs (O) and they are used to action the motor (the first one to a direction and the second one to the opposite direction) and their location is 0 for the first and 1 for the second. The third line refers to an input variable (I) (gives information of a sensor) and its location is 0.

Now that you have understood what those lines mean, you can configure the variables properly. So, you can create three global variables:

1. **Name:** AT1_MotorDirection1; **Type:** Global; **Location:** %QX0.0.1.0
2. **Name:** AT1_MotorDirection2; **Type:** Global; **Location:** %QX0.0.1.1
3. **Name:** AT1_Sensor; **Type:** Global; **Location:** %IX0.0.0.0

After this, you can instantiate those variables in the POU created. You just have to create the 3 following variables:

1. **Name:** AT1_MotorDirection1; **Type:** External
2. **Name:** AT1_MotorDirection2; **Type:** External
3. **Name:** AT1_Sensor; **Type:** External

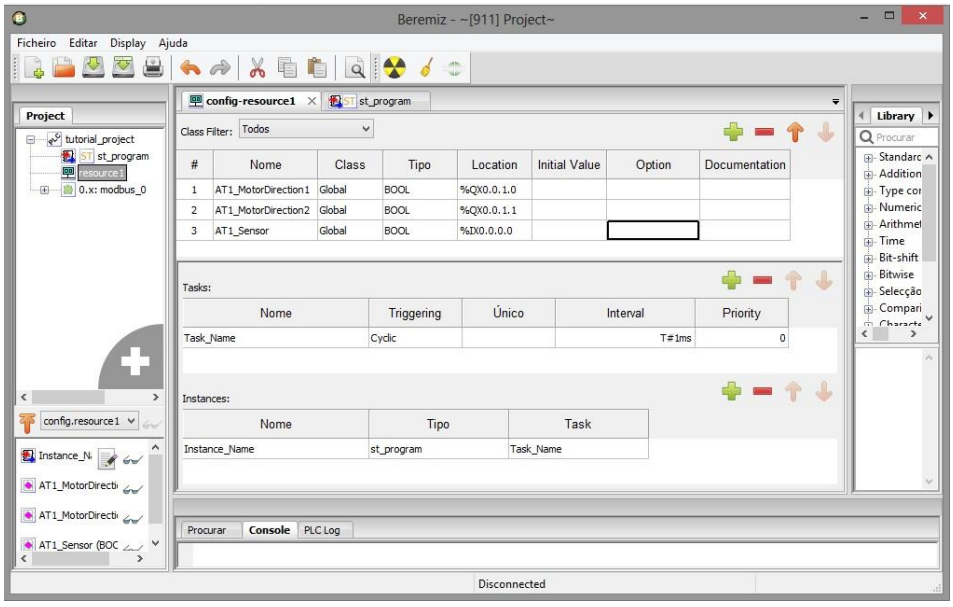

Figura 57 - Variables Configuration

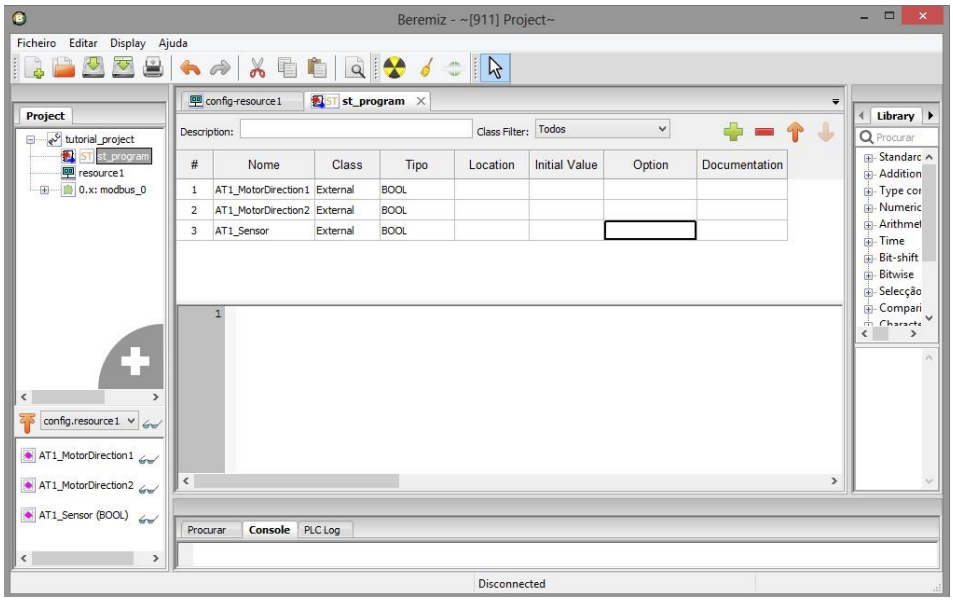

Figura 58 - Variables configuration (2)

Now, depending on what you want to do, you should configure the variables that you will use. With this example, you should be able to configure all the variables that are in the io.xls file. To develop the programs that we will show in this tutorial, I'll tell you what variables you should configure in the next chapters.

Developing Simple Programs

After reading the tutorial till this point, you should be able to create a POU, configure the communications protocol, add the variables that you need, create a new task and instantiate the program associated to the task you have created.

Now, the only thing left for you to know is how to write simple programs, compile them and transfer them to the PLC.

ST Mini Program

In first place, the language that will be demonstrated to you is the ST (Structured Text) language. Go back to the "Creating a POU" chapter and create a new POU. You can give the name that you want to the program. Define POU type as "program" and Language as "ST". To this program, we will use all AT1, ST1 and ST2 variables. So, to add these variables, go to the io.xls file and see what lines give you information about those variables. Check Figure 59 to see what you should see after adding those variables (in this tutorial, it will be used the method global/external variables instead of creating local variables in the ST program).

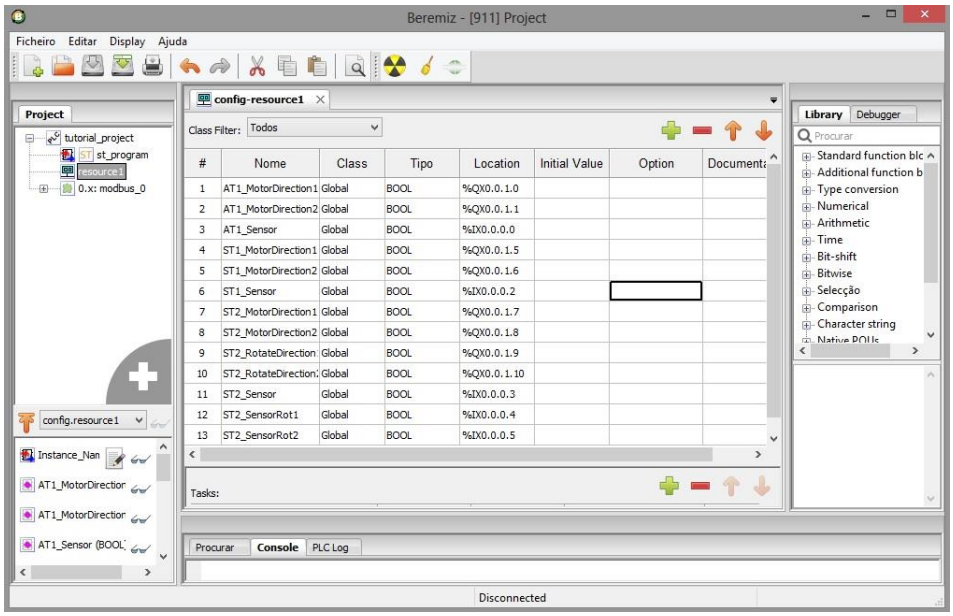

Figura 59 - Configuring ST Variables

Now that you have created the global variables in the resource and the external variables in the ST Program, you have to write the program. In your ST Program, you should be able to see a section that is there for you to write in Structured Text.

In this tutorial, this program will transport a piece from AT1 Warehouse to ST2 Rotator, passing through ST1 Conveyor. After the piece gets to ST2, this rotator should rotate 90 degrees, aligning with ST3 Machine.

So, in order to accomplish this objective, you should write the following instructions in your ST writing section:

```
IF (AT1_Sensor = TRUE) AND (ST1_Sensor = FALSE) THEN
  AT1_MotorDirection2 := TRUE;
  ST1_MotorDirection2  := TRUE;
END_IF;
IF (ST1_Sensor = TRUE) AND (ST2_Sensor = FALSE) THEN
  ST1_MotorDirection2:= TRUE;
  ST2_MotorDirection2 := TRUE;
END_IF;
IF (ST2_Sensor = TRUE) THEN
  ST2_RotateDirection2 := TRUE;
END_IF;
IF (ST2_SensorRot2 := TRUE) THEN
  ST2_RotateDirection2:= FALSE;
  ST2_MotorDirection2 := TRUE;
END_IF;
```

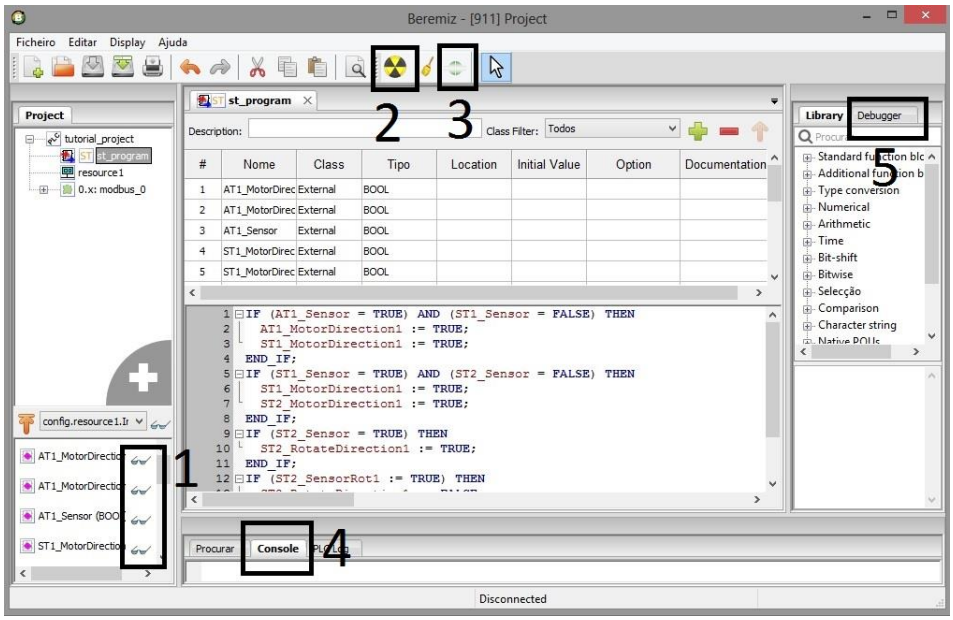

Figura 60 - ST Program Instructions

After writing these instructions, you should pay attention to Figure 60. The first thing that you have to do is clicking in the Build button to compile the program (marked as the rectangle 2 of Figure 60). You will get no errors, but if you made any mistake, the program will not compile and you will be able to see what is wrong by checking the console (rectangle 4 of Figure 60). Now that you have built your program, you can transfer it to the PLC. To do that, you should press the button 3 of the image above. The rectangle 1 is a set of glasses. Each glasses correspond to a variable (the variable name correspondent to the glasses is immediately at their left) and, if you click in the glasses, you are saying to Beremiz that you want to see the variable state in the Debugger menu (described as 5 in the Figure 60).

Now you have to transfer the program to the PLC and start running the program (check the images below).

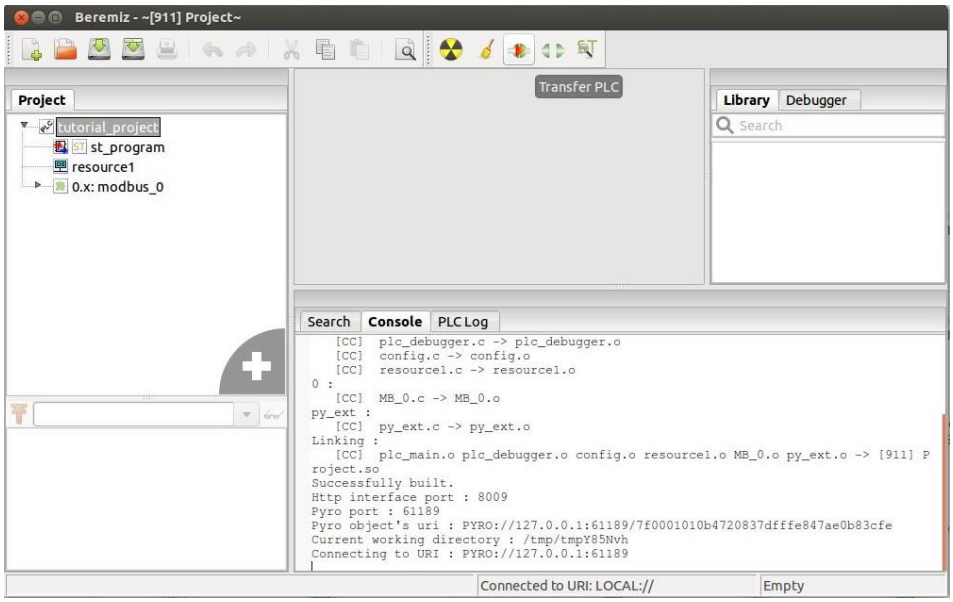

Figura 61 - Transfer to PLC

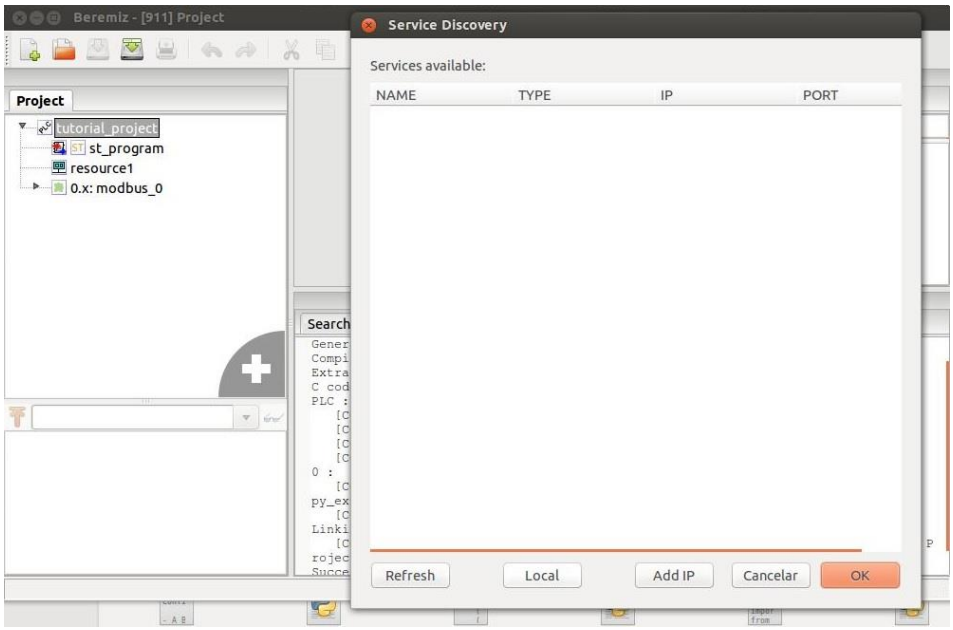

Figura 62 - Connect to PLC

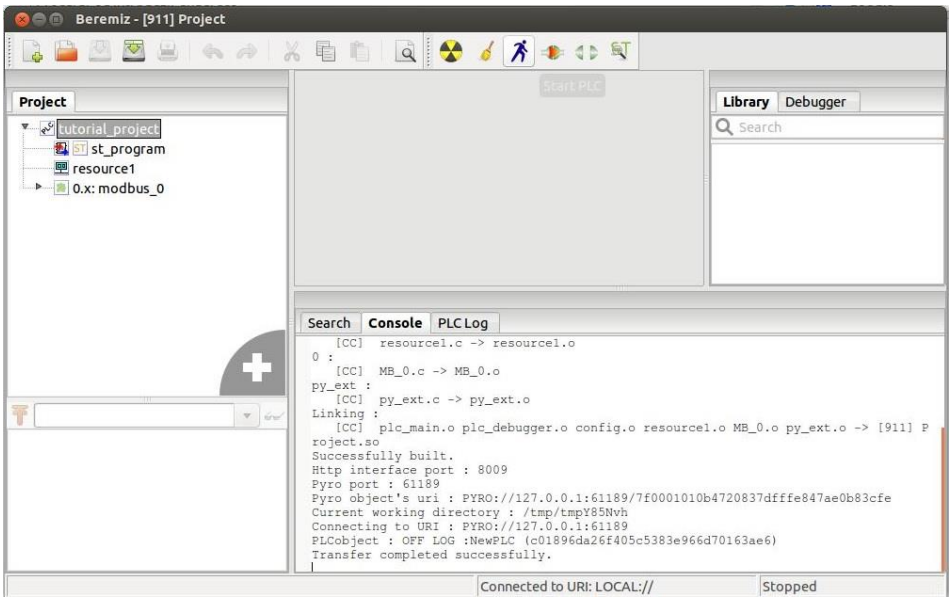

Figura 63 - Start PLC

Because we are using the localhost as a PLC, you have to click in local when you connect to the PLC (Figure 62).

To see what the program is doing, you have to open the simulator. The commands to open the simulator are one the Installing chapter, in the Simulator section. If you put your mouse at the top of any “rectangle” of the simulator, it will appear a label describing what that rectangle is. So, you should right click in your mouse in AT1 Warehouse (the rectangle at the left top of your simulator) and Add a Piece. Now, you should be able to see what is described as the objective of this program.

SFC Mini Program

Now that you have done your first ST Program, you can move to next step. In this chapter, you will do the exact same thing that you have done in the previous chapter but instead of writing the instructions in ST, you will write them in SFC (Sequence Flow Chart).

As we defined the variables as Global in the resource menu, now you just have to add a new POU, define *POU type* as “program” and *Language* as “SFC”. Now that you have a new program, you must add to this program the external variables correspondent to the global ones that you have created in the resource menu in the previous chapter.

In the Figure 64, you have some rectangles made so it can be explained to you how to use the basic tools of SFC in Beremiz. It is very intuitive! If you know a little about IEC61131-3, you know that in order to start, you need to create an Initial Step. The button to do that is represented in the Figure 64 with the number 1. The number 2 and 3 are the buttons to add a new step and a new transition, respectively. To add a new action, you should click in number 4. Divergences and Jumps are also possible, as everything of IEC61131-3 is. You can click in

your SFC section to add these elements too, as it is shown in Figure 64 and Figure 65. It is really simple.

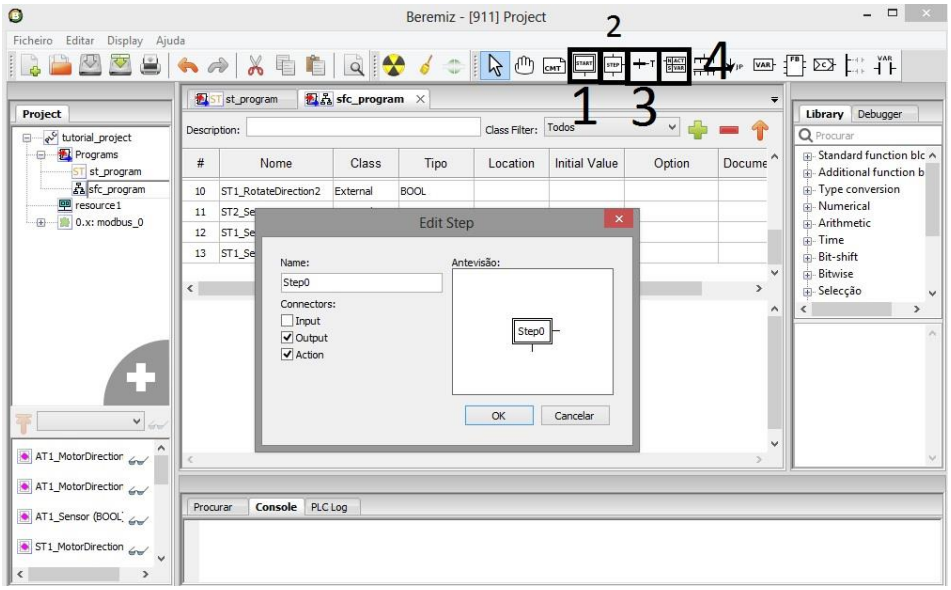

Figura 64 - SFC Program

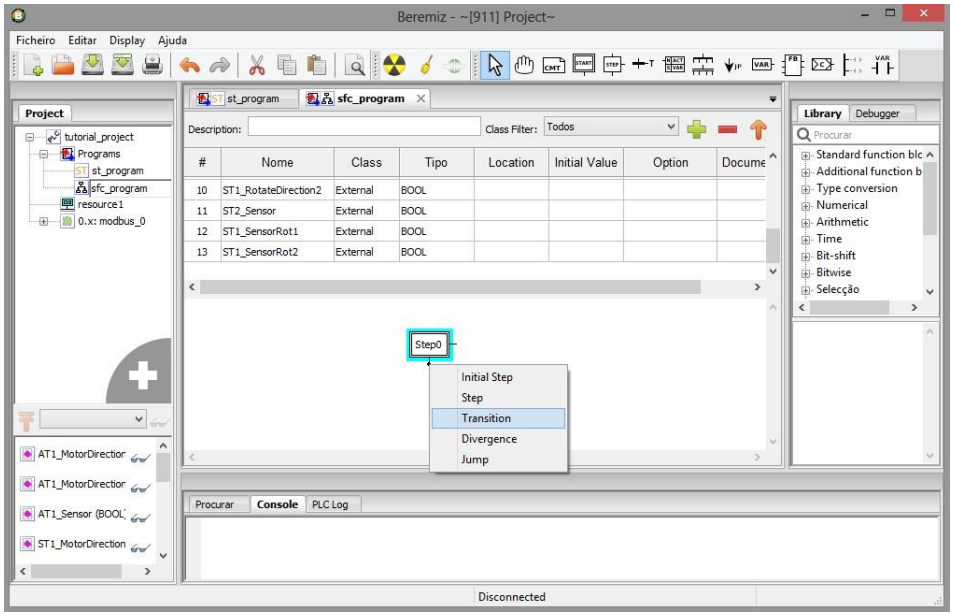

Figura 65 - Adding elements to a SFC program

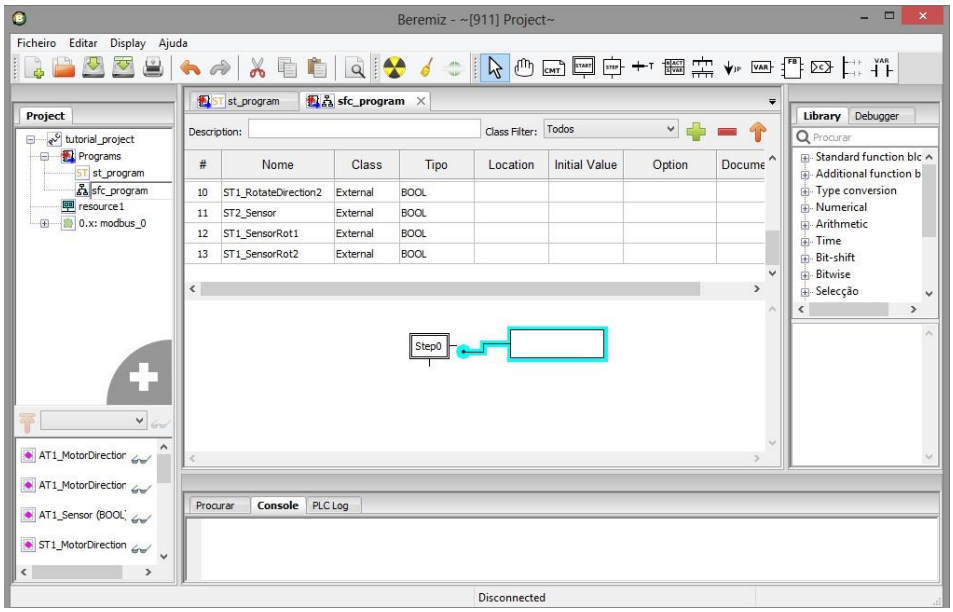

Figura 66 - Linking elements

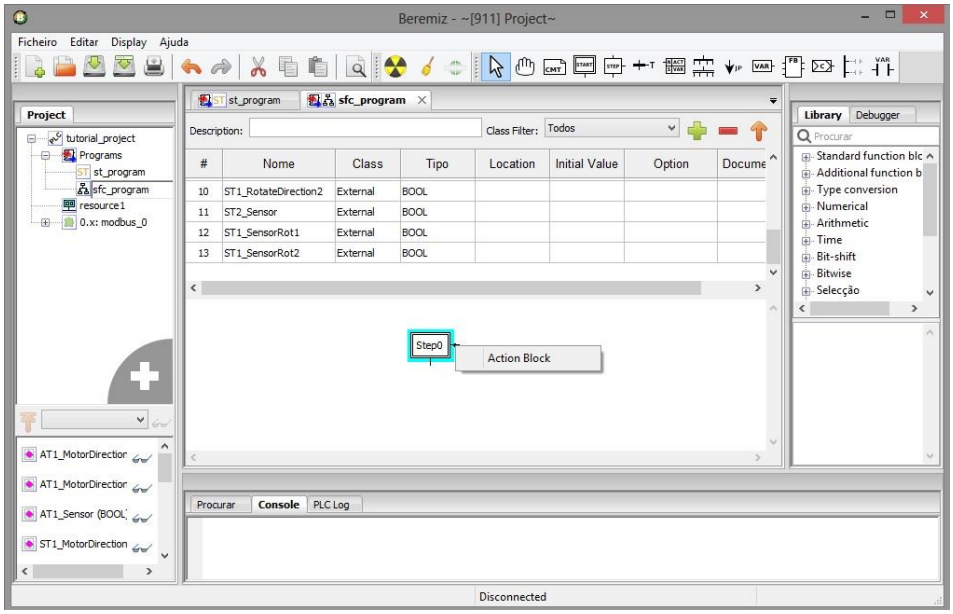

Figura 67 - Add action block

Now that you have read this little resume of the SFC tools, you should do follow the next instruction list:

1. Add an initial step. You can name it whatever you want. An initial step does not have an input, so select only Action and Output.
2. In step 1, you selected an action. So, now you need to add a new action block. In the action block you have to click in the green + to add a new action.
3. You only have to fill 3 fields, which are: **Qualifier** (choose N), **Tipo** (choose Action because it's the best choice to write multiple instructions and we will use it in the first action block) and **Value** (choose action0 – explained in the next steps).

4. In the Value option, you have a drill to choose one of the actions that are associated to that POU. But… you haven't created any action yet. So right click in your SFC program and **add a new action**. You can see this in Figure 68. Name it with action0 for example and define ST as the Language.
5. In the ST section of the action you just created **write the following content** to initialize all variables as "FALSE":

```
AT1_MotorDirection1 := FALSE;
AT1_MotorDirection2 := FALSE;
AT1_Sensor := FALSE;
ST1_MotorDirection1 := FALSE;
ST1_MotorDirection2 := FALSE;
ST1_Sensor := FALSE;
ST2_MotorDirection1 := FALSE;
ST2_MotorDirection2 := FALSE;
ST1_RotateDirection1 := FALSE;
ST1_RotateDirection2 := FALSE;
ST2_Sensor := FALSE;
ST2_SensorRot1 := FALSE;
ST2_SensorRot2 := FALSE;
```

6. Now that your action is completed, you can **select action0** as it was said in step 3.
7. You need a condition for your program to leave initial step to the next step so you need to **add a transition**.
8. In this transition, you will **choose the inline option and write** the following on the field: AT1_Sensor = TRUE AND AT2_SENSOR = FALSE; (see Figure 70).
9. We have to define now what will happen in the step0. So, you have to **add a new step**. This step is not like step0 because it needs an input, and output and an action.
10. In the action, we will now **choose Inline** instead of Action in the Value option. You have to add 2 actions that are the following:

    Qualifier: N; Tipo: Inline; **Value:** AT1_MotorDirection2 := TRUE;

    Qualifier: N; Tipo: Inline; **Value:** ST1_MotorDirection2 := TRUE;

    (look to Figure 70)
11. The following steps are very similar to these 10 first steps, so it will be explained to you not so detailed. So, **add a new transition**. In this transition you will write: ST1_Sensor = TRUE .

12. Now, add a **new step** with the 2 following actions associated:

    Qualifier: N; Tipo: Inline; **Value:** AT1_MotorDirection2 := FALSE;

    Qualifier: N; Tipo: Inline; **Value:** ST1_MotorDirection2 := FALSE;

13. After this step, you will **add a new transition** checking if there is any piece in ST2. If not, you can move the piece from ST1 to St2. So, to check if there's anything in ST2, write ST1_Sensor = TRUE AND ST2_Sensor = FALSE in the transition Inline field.
14. The step that you will add after the transition of step13 will order ST1 and ST2 to move in direction 2. So add the following actions to the action block:

    Qualifier: N; Tipo: Inline; **Value:** ST1_MotorDirection2 := TRUE;

    Qualifier: N; Tipo: Inline; **Value:** ST2_MotorDirection2 := TRUE;

15. The unique propose of the next transition/step is just to stop ST1 and ST2 from moving when there is no longer necessary. So, you should **add a new transition** with the condition: ST2_Sensor = TRUE .
16. Now, the next step should stop the motor of ST1 and ST2. Besides that, you should order ST2 to start rotating. In order to that, **write the following inline actions**:

    Qualifier: N; Tipo: Inline; **Value:** ST1_MotorDirection2 := FALSE;

    Qualifier: N; Tipo: Inline; **Value:** ST2_MotorDirection2 := FALSE;

    Qualifier: N; Tipo: Inline; **Value:** ST2_RotateDirection2 := TRUE;

17. Now, we only have to wait for the ST2_SensorRot2 turn TRUE, so we can order the rotator to stop rotating and start moving in direction 2. So, **add a transition** with ST2_SensorRot2 = TRUE.
18. In this final step we just have add the following two inline actions:

    Qualifier: N; Tipo: Inline; **Value:** ST2_RotateDirection2 := FALSE;

    Qualifier: N; Tipo: Inline; **Value:** ST2_MotorDirection2 := TRUE;

19. Now, you can add a transition only saying TRUE so we can go the next step. Instead of adding a new step, you can make a jump to step0. The program will work as a loop.

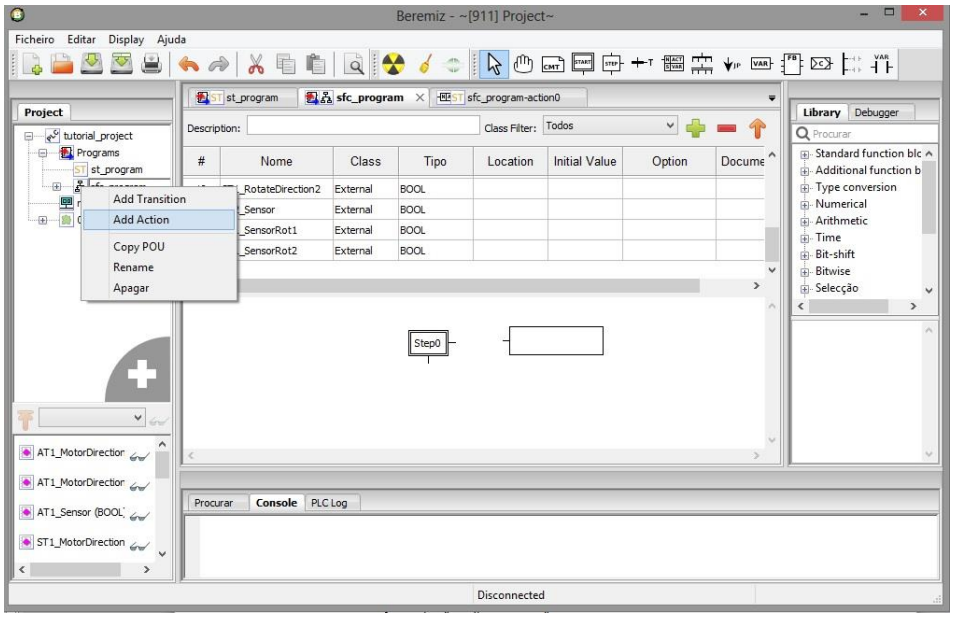

Figura 68 - Add action

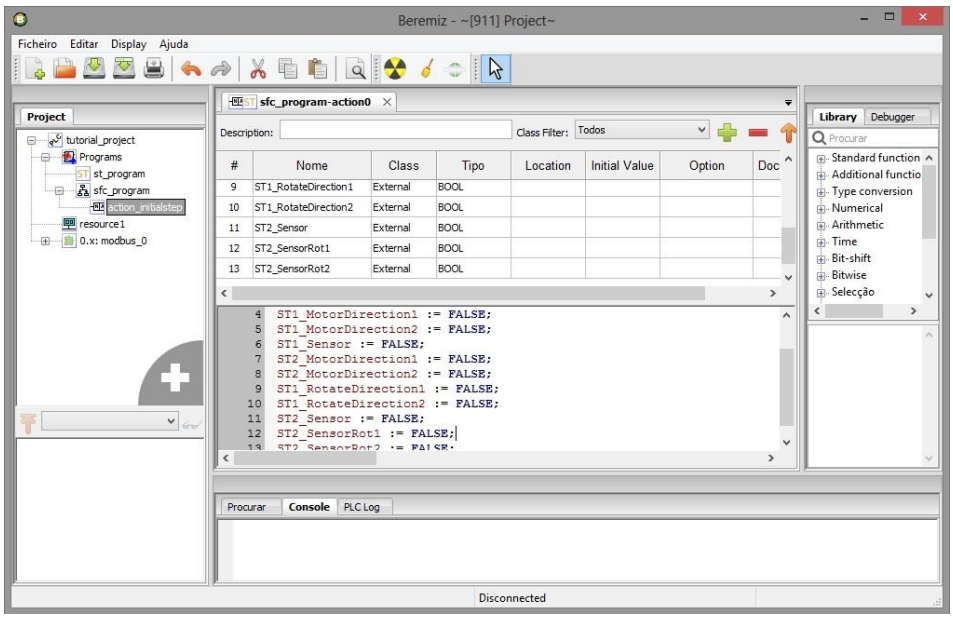

Figura 69 - Action content

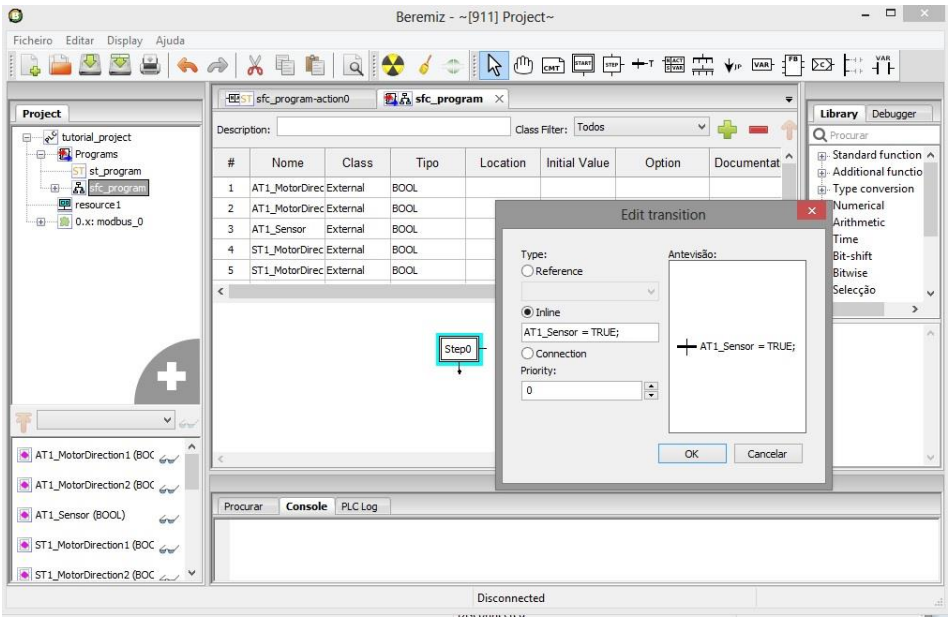

Figura 70 - Transition Inline

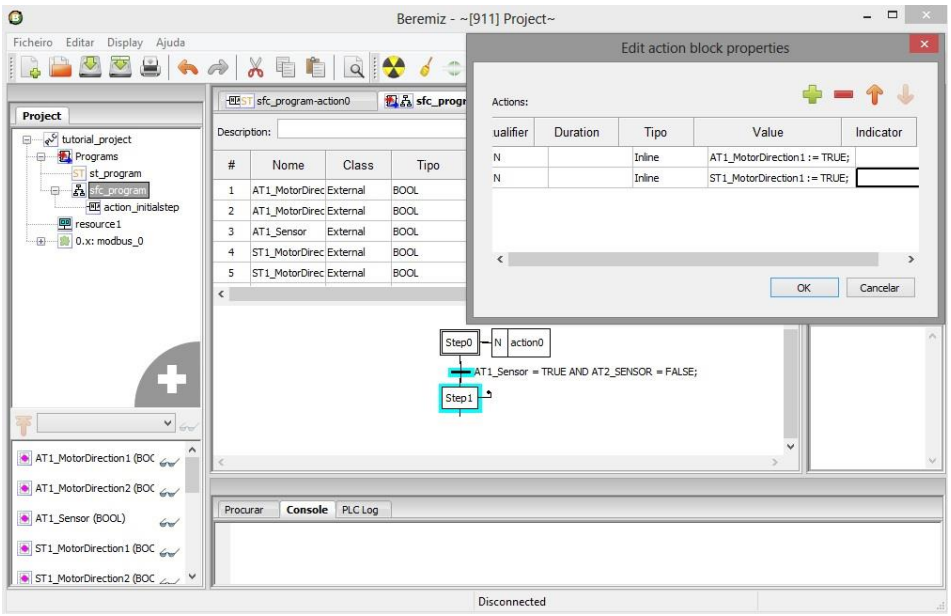

Figura 71 - Inline Action

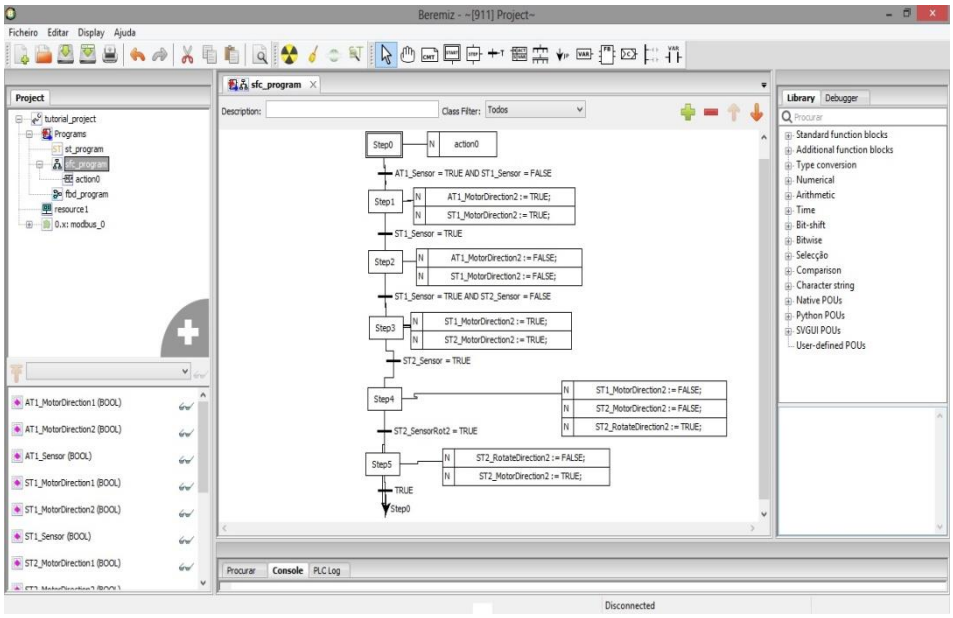

Figura 72 - SFC program

FBD Mini Program

In the demonstration of the Function Block Diagram, will not be used the program. Instead of that we will only show you how you can use Beremiz to program in this language.

After adding a new POU to your tutorial project, choose the FBD language. The procedure is exactly the same as explained before in ST Mini Program and SFC Mini Program.

Now that you have your new POU, when you right click in your FBD section, it will open a small box, where you can click to add a Block or a Variable. Then, all that you need to do is to link the elements that you add.

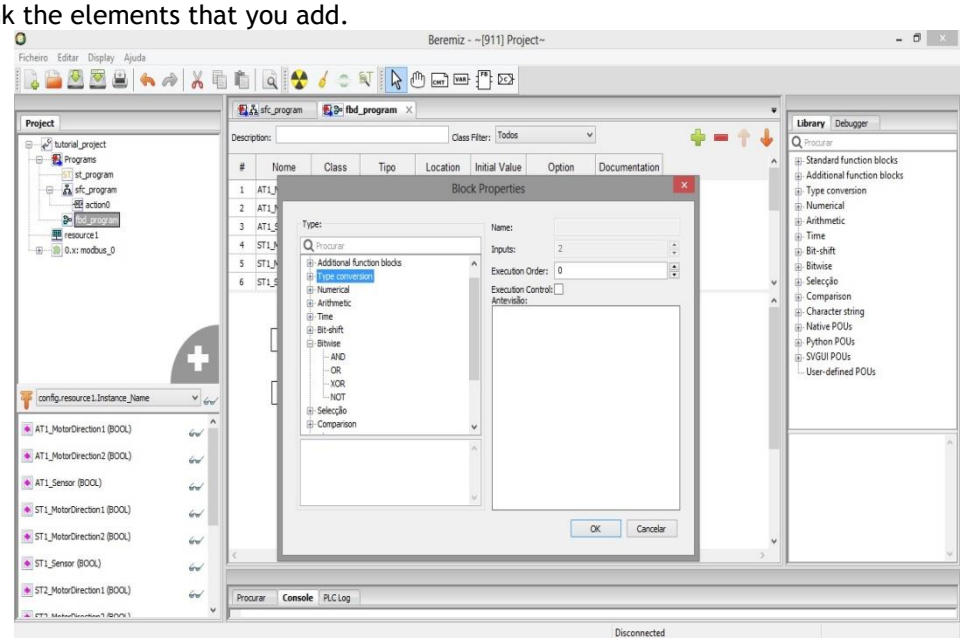

Figura 73 - Block Properties

When you add a new Block, it will appear a menu in your Beremiz view and you can search the list so you can add the block that you want. If you want to, you can create a new Function Block (with the same procedure that you have created the programs in ST, FBD and SFC) to do anything that you want. This Function Block will also appear in the list.

The program that was done for demonstration to you in this tutorial was the following:

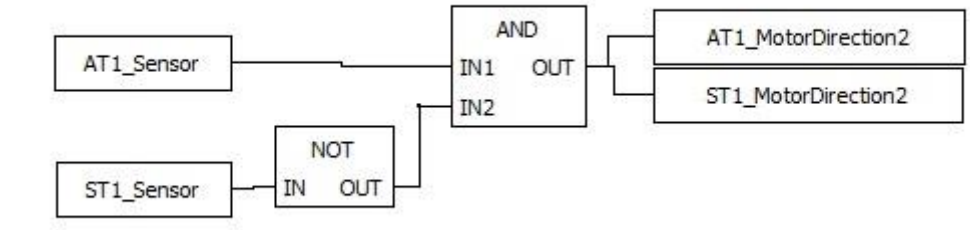

Figura 74 – Function Block Diagram

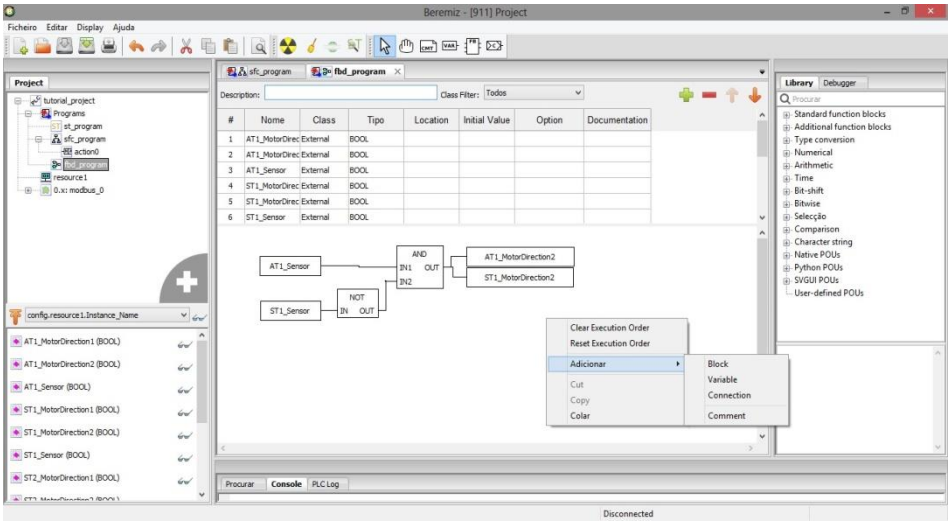

Figura 75 - Adding elements to a FBD program

We want a piece to go from AT1 to ST1, only if ST1 is free, obviously. So, we need to check if there is a piece is AT1 and there is not a piece in ST1 so we can order those motors to start moving.

In order to do that, you should add AT1_Sensor and ST1_Sensor variables. Then, we want to see if AT1_Sensor is **NOT** free **AND** ST1_Sensor is free. Everything that we need to do now is to add two blocks. An AND block and a NOT block. Link variable AT1_Sensor to one of the AND inputs (you can edit this block to add as many inputs as you want to) and the ST1_Sensor to the NOT input. Then, link the NOT output the second AND input. To end this program, you just have to link AT1_MotorDirection2 and ST2_MotorDirection2 to the AND output.

Final Notes

Thank you for downloading this tutorial! There are a lot more features available in Beremiz such as functions or Function Blocks that were not covered in this tutorial. If you want to know more about Beremiz, please visit www.beremiz.org.

Referências

[1] C. Neves, L. Duarte, N. Viana, e V. Ferreira. Os dez maiores desafios da automacao industrial: as perspectivas para o futuro. Em II Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educaçao Tecnológica, Joao Pessoa, Paraıba, Brasil, 2007

[2] A.A.B. Buccioli, E.R. Zorzal, e C. Kirner. Usando realidade virtual e aumentada na visualização da simulação de sistemas de automação industrial. Em SVR2006-VIII Symposium on Virtual Reality, 2006

[3] Andreas Ahlqvist, Conversion of SA30 PLC Program - Bachelor Thesis, 2013

[4] Karl-Heinz John, Michael Tiegelkamp, IEC 61131–3: Programming Industrial Automation Systems: Concepts And Programming Languages, Requirements for Programming Systems, AIDS to Decision-making Tools, 2001

[5] IEC 61131-3, 2nd Ed. Programmable Controllers - Programming Languages

[6] NATIONAL COMMUNICATIONS SYSTEM TECHNICAL INFORMATION BULLETIN 04-1 Supervisory Control and Data Acquisition (SCADA) Systems, October 2004

[7] “What is SCADA?” Disponível em http://www.inductiveautomation.com/what-is-scada [Consultado em Junho-2014]

[8] “SCADA screenshot” Disponível em http://www.windenergysolutions.nl/sites/default/files/SCADA%20screenshot.png [Consultado em Junho-2014]

[9] Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing

[10] “Free Software Definition” Disponível em http://www.gnu.org/philosophy/free-sw.html [Acesso Maio-2014]

[11] “Open Source Misses the Point” Disponível em https://www.gnu.org/philosophy/open-source-misses-the-point.html [Acesso Maio-2014]

[12] “Copyleft” Diponível em https://www.gnu.org/copyleft/copyleft.html [Acesso Maio-2014]

[13] “Microsoft joins MIT Kerberos Consortium” Disponível em http://newsoffice.mit.edu/2008/kerberos-microsoft-0331 [Acesso Junho-2014]

[14] “Escrevendo TAGS com PHP” Disponível em http://www.scadabr.com.br/?q=node/262 [Acesso Junho-2014]

[15] “GNU GPL” Disponível em http://www.gnu.org/copyleft/gpl.html [Acesso Maio-2014]

[16] “GNU LGPL” Disponível em http://www.gnu.org/copyleft/lgpl.html [Acesso Maio-2014]

[17] “GNU AGPL” Disponível em http://www.gnu.org/copyleft/agpl.html [Acesso Maio-2014]

[18] Vanessa Sabino, Fábio Kon, Licenças de Software Livre - História e Características - Relatório Técnico RT-MAC-IME-USP 2009-01 Departamento de Ciência da Computação – Relatorio Técnico, Instituto de Matemática e Estatística da Universidade de São Paulo, Março de 2009

[19] “Modbus Request and Response” Disponível em http://gridconnect.com/blog/wp-content/uploads/2013/03/MODBUS-Request-Response.bmp [Acesso Junho-2014]

[20] “Digital Rights Manager” Disponível em https://drm-pt.info/ [Acesso Junho-2014]

[21] Modbus-IDA. MODBUS Application Protocol Specification v1.1b, Dezembro de 2006

[22] Vasco Alexandre Martins das Neves Fernandes, ’Driver’ Modbus para Ambiente de Desenvolvimento IEC61131-3, Tese de Mestrado da Faculdade de Engenharia da Universidade do Porto, Julho de 2009

[23] Modbus-IDA. MODBUS Messaging on TCP/IP Implementation Guide v1.0b

[24] “Modbus ADU and PDU” Disponível em http://www.simplymodbus.ca/images/adu_pdu.PNG [Acesso Junho-2014]

[25] “Open OCS Automation Suite” Disponível em http://www.systec-electronic.com/en/products/automation-components/development-and-configuration-tools/openpcs-iec-61131-3-automation-suite

[26] “CODESYS software” Disponível em www.3s-software.com [Acesso Fevereiro-2014]

[27] Edouard Tisserant, Laurent Bessard, and Mário de Sousa. An Open Source IEC 61131-3 Integrated Development Environment. INDIN 2007, 2007

[28] “Beremiz” Disponível em www.beremiz.org [Acesso Fevereiro-2014]

[29] “MBLogic Project” Disponível em http://mblogic.sourceforge.net/ [Acesso Fevereiro-2014]

[30] “ClassicLadder” Disponível em http://sourceforge.net/projects/classicladder/ [Acesso Fevereiro-2014]

[31] “SoapBox Automation” Disponível em http://soapboxautomation.com/products/soapbox-snap/ [Acesso Fevereiro-2014]

[32] “Ladder Logic” Disponível em http://cq.cx/ladder.pl [Acesso Fevereiro-2014]

[33] “OLinuXino” Disponível em https://www.olimex.com/Products/OLinuXino/ [Acesso Fevereiro-2014]

[34] “OLinuXino” Disponível em “http://media.digikey.com/Photos/Olimex%20LTD/MFG_A13-OLINUXINO-WIFI.jpg [Acesso Fevereiro-2014]

[35] “Arduino” Disponível em http://arduino.cc/en/pub/skins/arduinoWide_SSO/slider_home/h_01.jpg [Acesso Fevereiro-2014]

[36] “Raspberry PI” Disponível em http://www.adafruit.com/images/1200x900/998-00.jpg [Acesso Fevereiro-2014]

[37] “Projecto Open-PLC” Disponível em https://code.google.com/p/open-plc/wiki/Main [Acesso Fevereiro-2014]

[38] ”openSCADA Project” Disponível em http://oscada.org/ [Acesso Fevereiro-2014] e http://www.openscada.org [Acesso Fevereiro-2014]

[39] “Mango Project” Disponível em http://mango.serotoninsoftware.com/home.jsp [Acesso Fevereiro-2014]

[40] “ScadaBR Software” Disponível em http://www.scadabr.com.br/ [Acesso Fevereiro-2014]

[41] “Proview~Open Source Project Control” Disponível em http://www.proview.se/v3/ [Acesso Fevereiro-2014]

[42] “Szarp Open Source SCADA Software” Disponível em http://www.szarp.org/ [Acesso Fevereiro-2014]

[43] “ PVBrowser Simple Process Visualization“ Disponível em http://pvbrowser.de/pvbrowser/index.php?lang=en&menu=1/ [Acesso Fevereiro-2014]

[44] “What is Likindoy?” Disponível em http://www.likindoy.org/en/About-Likindoy/what_is_it [Acesso Fevereiro-2014]

[45] “PascalSCADA HMI/SCADA for Developers” Disponível em http://pascalscada.blogspot.pt/ [Acesso Fevereiro-2014]

[46] “Parijat HMI/SCADA Open-Source Development System” Disponível em http://www.parijat.com/HMI_SCADA_System.aspx/ [Acesso Fevereiro-2014]

[47] “SCADA Open Source – SOS” Disponível em http://scada-open-source-sos.soft112.com/ [Acesso Fevereiro-2014]

[48] “VISUAL web-enabled open source SCADA and HMI software for Linux” Disponível em http://visual.sourceforge.net/new/index.php [Acesso Fevereiro-2014]

[49] “Open Web SCADA - open source SCADA like system based on the web solutions” Disponível em https://code.google.com/p/openwebscada/ [Acesso Fevereiro-2014]

[50] “FreeScada Project” Disponível em http://sourceforge.net/projects/free-scada [Acesso Fevereiro-2014]

[51] “Simple Extensive Automation” Disponivel em http://sea.sourceforge.net/ [Acesso Fevereiro-2014]

[52] “Python Programming Language”. Disponivel em http://www.python.org/ [Acesso Fevereiro-2014]

[53] “The Making of Python” Disponível em http://www.artima.com/intv/pythonP.html [Acesso Fevereiro-2014]

[54] “Python bindings to the wxWidgets cross platform toolkit” Disponível em http://www.wxpython.org/ [Acesso Fevereiro-2014]

[55] “WxWidgets. A C++ cross-platform GUI library” Disponível em http://www.wxwidgets.org/. Acesso Fevereiro-2014]

[56] PLCopen Technical Committee 6. XML Formats for IEC 61131-3, v2.01. Technical report, PLCOpen, 2009 [Acesso Fevereiro-2014]

[57] “PLCOpen TC6 XML Official Schema” Disponível em http://www.plcopen.org/tc6_xml/tc6_ xml_v201.xsd [Acesso Junho-2014]

[58] Daniel Andre da Silva Petim Batista, Automação de linha de fabrico flexível do DEEC. Tese de mestrado, Faculdade de Engenharia da Universidade do Porto, 2000

[59] Beremiz Manual - The Free and Open Source IEC 61131-3 Automation IDE

[60] “PVBrowser’s Home Project” Disponível em https://build.opensuse.org/project/show/home:pvbrowser [Acesso Fevereiro-2014]

[61] “PVBrowser“ Disponível em http://www.flussiliberi.it/wp-content/uploads/2010/08/pvbRun1.jpeg [Acesso Maio-2014]

[62] "Eclipse Public License - v 1.0", lançado em 2006-09-12.

[63] “Eclipse SCADA Project” Disponível em http://www.eclipse.org/eclipsescada/ [Acesso Maio-2014]

[64] Jens Reimann, Jürgen Rose, Eclipse SCADA – The Definite Guide, 31 de Março de 2014

[65] IBH Systems, “Eclipse SCADA – an introduction” Disponível em http://www.eclipsecon.org/europe2013/sites/eclipsecon.org.europe2013/files/Eclipse%20SCADA%20V1.pdf [Acesso Junho-2014]

[66] “Representação gráfica ScadaBR” Disponível em http://ceticsc.files.wordpress.com/2010/02/nota_2_1.jpg [Acesso Junho-2014]

[67] “Application Program Interface definition” http://foldoc.org/Application+Program+Interface [Acesso Junho-2014]

[68] “SOAP Introduction” Disponível em http://www.w3schools.com/webservices/ws_soap_intro.asp [Acesso Junho-2014]

[69] “Minicursos ScadaBR” Disponível em https://sites.google.com/a/certi.org.br/certi_scadabr/home/minicursos/scadabr [Acesso Maio-2014]

[70] Nuno Fortunato Oliveira, Automação de Linha de Produçao Flexivel, Tese de Mestrado da Faculdade de Engenharia da Universidade do Porto, Fevereiro 2013

[71] “Documento de Referência do ScadaBR” Disponível em https://sites.google.com/a/certi.org.br/certi_scadabr/requisitos [Acesso Maio-2014]

[72] “Representação gráfica ScadaBR” Disponível em http://www.scadabr.org.br/sites/default/files/builder.png [Acesso Maio-2014]

[73] “Python and ScadaBR API” Disponível em http://www.scadabr.org.br/?q=node/443 [Acesso Junho-2014]

[74] “Autenticação Web Cliente Service” Disponível em http://www.scadabr.org.br/?q=node/652 [Acesso Junho-2014]

[75] “A licença de Documentação Livre do GNU” Disponível em https://www.gnu.org/licenses/licenses.html#FDL [Acesso Junho-2014]